(19) Europäisches Patentamt
European Patent Office
Office européen des brevets

(11) EP 0 871 303 A2

(12) EUROPEAN PATENT APPLICATION

(43) Date of publication:
14.10.1998 Bulletin 1998/42

(51) Int. Cl.^{6}: H04L 1/00, H04L 25/49

(21) Application number: 98106516.2

(22) Date of filing: 08.04.1998

(84) Designated Contracting States:
AT BE CH CY DE DK ES FI FR GB GR IE IT LI LU MC NL PT SE
Designated Extension States:
AL LT LV MK RO SI

(30) Priority: 08.04.1997 US 43066 P

(71) Applicants:
- Demjanenko, Victor
Pendleton, New York 14094 (US)
- Hirzel, Frederic J.
Sunnyvale, California 94086 (US)

(72) Inventors:
- Demjanenko, Victor
Pendleton, New York 14094 (US)
- Hirzel, Frederic J.
Sunnyvale, California 94086 (US)

(74) Representative: Celestino, Marco
ABM, Agenzia Brevetti & Marchi,
Via A. Della Spina 40
56125 Pisa (IT)

(54) Method for determining the attenuation of a PCM signal over a digital channel

(57) A method of determining digital channel attenuation; comprising the steps of: receiving a known training sequence of PCM codes, which PCM codes are subjected to the attenuation within the digital channel; quantizing the received known training sequence of PCM codes according to a predetermined thresholding procedure; identifying identical PCM codes created as a result of the thresholding procedure; and, determining the attenuation of the digital channel based upon the identification of identical PCM codes. A method is also disclosed for determining a digital channel PCM code transformation comprising receiving a known training sequence of PCM codes, which PCM codes are subjected to the PCM code transformation within the digital channel, quantizing the received known training sequence of PCM codes according to a predetermined thresholding procedure, and determining the transformation of transmitted codes to those received . A method is also disclosed for improved echo cancellation in a communications network having an analog and a digital modem, comprising saving codes transmitted from the digital modem to the analog modem for echo cancellation, transforming, by a mapping table, codes transmitted from said digital modem to codes received by the analog modem, and, using the received codes as a reference signal for cancellation of echo. A method of improved spectral shaping using a transmit shaping transfer function in a communications network having an analog and a digital modem, comprising, transforming, by a mapping table, codes transmitted from the digital modem to codes received by the analog modem, using the received codes for transformation to their linear value equivalent representations, and, applying the linear value representations to the transmit shaping transfer function.

EP 0 871 303 A2

Description

Field of the Invention

This invention pertains generally to modem technology, more specifically to PCM modem technology, and more specifically, to a method for the discovery of the digital attenuation from a set of received PCM samples corresponding to a known set of transmit PCM samples.

Background of the Invention

This invention provides a method for the discovery of the digital attenuation from set of received pulse code modulation (PCM) samples corresponding to a known set of transmit PCM samples. The principal feature set used is the knowledge of which received PCM codes have become indistinguishable (i.e., are identical) as a result of the attenuation mapping. An alternative but equivalent feature set would be use of the absent received PCM codes.

The question of digital attenuation PAD mapping has been raised at recent meetings of the' Telephone Industry of America (TIA) PCM Modem Ad-hoc Committee. Many participants have been hoping for a single industry standard mapping solution. We provide a solution, which will allow PCM modems to function properly throughout the entire telephone network including with private telephone equipment (PBX's and key systems).

TIA 464A

TIA 464A is the industry standard plan for private telephone equipment. Among other recommendations, this specification provides the recommended losses between T1 lines and other devices. If a PBX is designed according to the loss-plan specification as outlined in sections 4.8.4 and 4.8.5 of 464A, then it should not be necessary to attenuate the PCM on incoming T1 lines for on-premise (ONS) connections, since the recommended 3-dB loss is typically implemented in the analog circuitry of the ONS codec.

However, Inter-Tel's experience with customers using T1 lines suggests that the recommended 3-dB of insertion loss is inadequate and that T1 attenuation pads are indeed necessary. While the standard -3 and -6 dB pads are usually adequate, Inter-Tel has provided a wider-range of insertion loss as options for PBX customers. Inter-Tel has implemented digital gain control on its T1 line cards using two different methods:

- ROM-based mapping
- DSP-based algorithm

ROM Based Mapping

Inter-Tel first implemented attention pads via an EPROM circuit, which used the incoming PCM code as 8 bits of an address and additional address bits selecting the attenuation in 1 dB increments. The output of the EPROM was the attenuated digital PCM code. In this implementation, the attenuation range covered from 0 dB to -12 dB in 1 dB steps.

Experience with customers shows that 0 to -6 dB seems to be the most-often used range of T1 pad values. Taking into account the fixed 3-dB of loss included with our ONS card, the net insertion loss for to Digital CO connection is then -3dB to -9dB, which mimics well the typical net losses experienced by customers on ONS to Analog CO connections.

All attenuation tables in the EPROM preserved the LSB of the PCM to preserve the state of the Robbed Bit Signaling (RBS), if present. However, the preservation of the LSB is unnecessary if no RBS information passes through the ROM-based gain control circuitry.

The ROM-based look-up tables were generated by a custom C program using the following algorithm:

1. Expand incoming 8-bit,u-Law PCM to corresponding 14-bit, signed integer linear value, x
2. Calculate output, y, based on equation: $y=INT\ [x*10^{(gain/20)})+0.500]$
3. Compress y back into its corresponding 8-bit u-Law value using G.711 *decision values*.

In addition to performing u-Law to u-Law digital gain control (DGC) through look-up tables, the EPROM also performs DGC through look-up tables for the following compression schemes:

- u-Law-to-A-law
- A-law-to-u-Law
- A-law-to-A-law

DSP Based Mapping

With the advent of DSP's, for their greater flexibility, the latest implementation uses the Analog Devices 21xx fixed point family of processors. The PCM codes are converted to/from linear values by the ADSP-21xx internal companing hardware. Inter-Tel's algorithm normalizes the linear value before applying the attenuation multiplier. The attenuation is selectable as a linear 1.15 fractional multiplier. The user interface selects the attenuation in fixed, 1-dB steps over a similar range as the previous implementation. The user configured dB attenuation is converted to the linear fractional multiplier by an algorithm. Moreover, the instructions in the Analog Devices DSP can use unbiased rounding, truncation and in newer family members, biased rounding. Inter-Tel's implementation uses the unbiased rounding option (RND) during the MAC instruction. This flexibility in rounding and truncation obviously can have a substantial effect on the actual attenuation mapping, with the additional caveat of having potentially different mappings for positive and negative PCM codes.

Detailed Description of the Preferred Embodiment

Detection of the Feature Set

After a receiver's equalizer has been trained, a known transmit sequence of PCM codes can be sent and received using either all or a useful subset of PCM codes. (This sequence may need to be sent multiple times to cover all 6 or 12 bit positions in a robbed bit signaling system.) The receiver can save the linear values for each received PCM code. From the known step size between adjacent PCM codes, the receiver can determine those PCM samples that it received which are indistinguishable. The indistinguishable samples arise from two transmit PCM codes which when scaled by the same attenuation and quantized according to a thresholding procedure create identical PCM codes.

An example taken from the attached attenuation table (pages A1-A2) which shows the attenuation for each uLaw PCM code when it is transformed by shifting from 1 to 26 codes numerically.

Example:

Transmitted		Received shifted by 1 PCM code			
uLaw	Linear	uLaw	Threshold	Attenuation	dB
129	7775	129	7647	0.983537	-0.14419
128	8031		7903	0.984062	-0.13955

Processing of Indistinguishable PCM Codes

The processing of the feature set relies on knowledge of the coding law (uLaw or Alaw) used. In practice, it is necessary to consider the preciseness of the numerics used in the attenuation process. Suggestions for accommodating implementation deviations is discussed in a later section. The attached C language source code provided as part of this application (pages B1- B21) implements a demonstration of the identification procedure indistinguishably codes. The basic algorithm is the following:

1. For each indistinguishable PCM code received, the original pair of transmitted PCM codes can be determined since the PCM codes are sent in a known training sequence. A set of indistinguishable transmitted PCM code pairs are determined from this training sequence.
2. For the first indistinguishable PCM code pair, determine the minimum attenuation required for the larger transmitted PCM code to be attenuated to a number of successive lower PCM codes. (A set of 30 successive lower PCM codes would be sufficient to cover an attenuation range of approximately 11 dB.) Then determine the maximum attenuation permitted for the smaller transmitted PCM code to be attenuated to the same successive lower PCM codes. (Minimum attenuation is numerically closer to one, i.e., a lesser attenuation, while a maximum attenuation is a greater attenuation.) If the maximum attenuation is a smaller attenuation than the minimum attenuation, discard the attenuation range as it represents a missing code rather than an indistinguishable code. The minimum and maximum attenuations for each successive lower PCM code forms an initial set of candidate attenuation ranges.
3. For each successive indistinguishable transmit PCM code pair, another set of possible attenuation ranges is. determined using the procedure as described in Step 2. Each element of the candidate attenuation range set cre-

ated in Step 2 is searched for either full or partial overlap with any element of the possible attenuation range set. (In case of partial overlap, the attenuation range in the candidate set can be reduced for refined attenuation accuracy.) In case of no overlap, the candidate attenuation range should be discarded. The discard occurs because two different indistinguishable codes must arise from the same linear transformation.

4. Step 3 is repeated for successive pairs of indistinguishable transmit PCM code pairs until either a) the set of candidate attenuation ranges is reduced to a single range or b) all the pairs of indistinguishable transmit PCM code pairs have been processed.

In the first case, the actual attenuation is bounded by the single remaining attenuation range. We can say it is the detected attenuation can be the average of the high and low attenuations. Alternatively, if the history of each overlapping attenuation range is maintained (or regenerated) a probabilistic approach of determining a weighted median value for determining the detected attenuation can be implemented.

For the second case where the attenuation appears to remains non-unique, the set of candidate attenuation ranges can be used to determine multiple sets of indistinguishable PCM code pairs can be determined for each candidate attenuation. The generated sets of indistinguishable PCM code pairs can then be matched with the original received set. The generated set using the correct attenuation should match nearly identically if not identically. The sets corresponding to incorrect potential attenuations would generate significant number of additional pairs of indistinguishable PCM code pairs or have missing indistinguishable PCM code pairs. Thus the correct detected attenuation can still be determined using this procedure. (Multiple candidate attenuations appear to arise from attenuations of 6 dB or greater combined with the decreasing step sizes used in with uLaw or Alaw companding.)

Processing of Missing PCM Codes

The procedure described above for using indistinguishable PCM codes to determine the attenuation can be adapted to operate with missing received PCM codes. The transmitted PCM codes corresponding each of the received PCM codes around the missing PCM code can be determined. Follow the same algorithm except that the minimum and maximum attenuations are exchanged in the algorithms. In other words, the larger transmitted PCM code produces the maximum attenuation and the smaller transmitted PCM code produces the minimum attenuation. With this adaptation, the algorithm described remains essentially the same.

Example:

Transmitted		Recieved shifted by I PCM code			
uLaw	Linear	uLaw	Threshold	Attenuation	dB
144	3999	144	3935	0.983996	-0.14013
143	4191		4063	0.969458	-0.26942

Other Variations

For robustness with real signals corrupted by noise, it may be desirable to allow a certain number of pairs of indistinguishable PCM codes not to match a candidate range. An alternative to simply dropping a candidate attenuation range would be to use a probabilistic greatest likelihood model for selection of the detected attenuation.

Furthermore, given that implementers of digital attenuation pads may understand and implement the mapping process slightly differently (i.e., use of G.711 threshold values or average between PCM codes) and allow various inaccuracies in the numerical operations (numeric representation, rounding, truncation, etc.), each candidate attenuation range may need to be broadened to make overlap again more likely. The broadening can be by a fixed value or a relative amount about the attenuation computed. The previous suggestion of discarding a certain number of pairs of indistinguishable PCM codes also addresses this problem as very few pairs are likely to be affected by these numerical inaccuracies.

As for digital attenuation pads that pass through the robbed bit signaling (RBS) information, the algorithms presented can also be applied except that `the indistinguishable transmit PCM code pairs will be separated by one code and a second indistinguishable transmit PCM code pair will adjacent. The candidate attenuation ranges become bigger, but still can be matched by the procedures presented in this invention.

The procedures described in this invention suggest a robust manner to identify the digital channel attenuation

though the use of received PCM codes corresponding to a known transmit PCM code sequence. The use of such an algorithm would be necessary when the FCC permits the channel power to be increased in compensation for the systematic digital attenuations.

We further suggest that knowledge of the exact received PCM code set may be preferred by both the analog and digital PCM modems. Echo cancellation in the digital PCM modem may operate with incrementally better performance with the knowledge of actual PCM codes presented to the Codec. The receiver's desired spectral shaping can be more precisely honored by having the transmitter use the knowledge of the actual PCM codes presented to the Codec in the shaping function implementation, then reversely mapping the PCM code after shaping to the transmit PCM code to be sent.

The exclusive use of pre-identified attenuation table mappings would make the PCM modem technology incapable of operating with existing private telephone equipment. It is recommended that the attenuation pad mappings be discovered for each connection.

Claims

1. A method of determining digital channel attenuation,comprising:

    receiving a known training sequence of PCM codes, which PCM codes are subjected to said attenuation within said digital channel;
    quantizing said received known training sequence of PCM codes according to a predetermined thresholding procedure;
    identifying identical PCM codes created as a result of said thresholding procedure; and,
    determining said attenuation of the digital channel based upon said identification of identical PCM codes.

2. A method of determining digital channel attenuation, comprising:

    receiving a known training sequence of PCM codes, which PCM codes are subjected to said attenuation within said digital channel;
    quantizing said received known training sequence of PCM codes according to a predetermined thresholding procedure;
    identifying PCM codes omitted as a result of said thresholding procedure; and,
    determining said attenuation of the digital channel based upon said identification of omitted PCM codes.

3. A method of determining a digital channel PCM code transformation comprising:

    receiving a known training sequence of PCM codes, which PCM codes are subjected to said PCM code transformation within said digital channel;
    quantizing said received known training sequence of PCM codes according to a predetermined thresholding procedure; and
    determining the transformation of transmitted codes to those received.

4. A method of improved echo cancellation in a communication network having an analog and a digital modem,comprising:

    saving codes transmitted from said digital modem to said analog modem for echo cancellation;
    transforming, by a mapping table, codes transmitted from said digital modem to codes received by said analog modem; and
    using said received codes as a reference signal for cancellation of echo.

5. A method of improved spectral shaping using a transmit shaping transfer function in a communications network having an analog and a digital modem, comprising:

    transforming, by a mapping table, codes transmitted from said digital modem to codes received by said analog modem;
    using said received codes for transformation to their linear value equivalent representations; and,
    applying said linear value representations to said transmit shaping transfer function.

Attenuation (in dB) required to shift to successive PCM codes

→ Creates a missing PCM code in the sequence received

← Creates a pair of indistinguishable PCM codes

A1

uLaw	Linear	by 1	by 2	by 3	by 4	by 5	by 6	by 7	by 8	by 9	by 10	by 11	by 12	by 13	by 14	by 15	by 16	by 17	by 18	by 19	by 20	by 21	by 22	by 23	by 24	by 25	by 26
255	0																										
254	2	-6,0206																									
253	4	-2,49877	-12,0412																								
252	6	-1,58362	-6,0206	-15,563																							
251	8	-1,15884	-4,0824	-8,51937	-18,0618																						
250	10	-0,91515	-3,09804	-6,0206	-10,4576	-20																					
249	12	-0,75577	-2,49877	-4,68166	-7,60422	-12,0412	-21,5836																				
248	14	-0,64369	-2,09471	-3,83771	-6,0206	-8,94316	-13,3801	-22,9226																			
247	16	-0,56057	-1,80353	-3,25456	-4,99756	-7,16044	-10,103	-14,54	-24,0824																		
246	18	-0,49647	-1,58362	-2,82668	-4,2776	-6,0206	-8,20349	-11,1261	-15,563	-26,1065																	
245	20	-0,44653	-1,41162	-2,49877	-3,74173	-5,19275	-6,93575	-9,11864	-12,0412	-16,4782	-26,0206																
244	22	-0,40407	-1,27338	-2,23948	-3,32663	-4,56959	-6,0206	-7,7636	-9,94549	-12,8491	-17,306	-26,8485															
243	24	-0,36967	-1,15984	-2,02915	-2,99525	-4,0824	-5,32536	-6,77637	-8,51937	-10,7023	-13,6248	-18,0618	-27,6042														
242	26	-0,34067	-1,06491	-1,85608	-2,72439	-3,69049	-4,77764	-6,0206	-7,47161	-9,21462	-11,3976	-14,3201	-18,767	-28,2995													
241	28	-0,31589	-0,98436	-1,7086	-2,49877	-3,36809	-4,33418	-5,42134	-6,66429	-8,11531	-9,85831	-12,0412	-14,9638	-19,4007	-28,9432												
240	30	-0,29447	-0,91515	-1,58362	-2,30787	-3,09804	-3,96735	-4,93345	-6,0206	-7,26356	-8,71467	-10,4576	-12,6405	-15,563	-20	-29,5424											
239	33	-0,54304	-1,12732	-1,743	-2,41148	-3,13572	-3,92589	-4,79521	-5,7613	-6,84845	-8,09141	-9,54243	-11,2854	-13,4683	-16,3909	-20,8279	-30,3703										
238	37	-0,48267	-1,5368	-2,11607	-2,73676	-3,40523	-4,12948	-4,91965	-5,78896	-6,75506	-7,84221	-9,08517	-10,5362	-12,2792	-14,4621	-17,3846	-21,8216	-31,364									
237	41	-0,43438	-1,37432	-2,42844	-3,00772	-3,6284	-4,29688	-5,02112	-5,81129	-6,68061	-7,6467	-8,73385	-9,97681	-11,4278	-13,1708	-15,3537	-18,2763	-22,7133	-32,2657								
236	45	-0,39488	-1,24296	-2,18289	-3,23702	-3,81629	-4,43697	-5,10545	-5,82969	-6,61986	-7,48918	-8,45627	-9,64243	-10,7854	-12,2364	-13,9794	-16,1623	-19,0849	-23,5218	-33,0643							
235	49	-0,36196	-1,13465	-1,98263	-2,92256	-3,97669	-4,55596	-5,17665	-5,84512	-6,56936	-7,35964	-8,22665	-9,19494	-10,2821	-11,5251	-12,9761	-14,7191	-16,902	-19,8246	-24,2616	-33,8039						
234	53	-0,33411	-1,04356	-1,81615	-2,66423	-3,60416	-4,65828	-5,23756	-5,85524	-6,52672	-7,25096	-8,04113	-8,81045	-9,67654	-10,8637	-12,2067	-13,8577	-15,4007	-17,5838	-20,5061	-24,9431	-34,4655					
233	57	-0,31024	-0,96609	-1,67654	-2,44813	-3,2982	-4,23614	-5,29026	-5,86954	-6,49022	-7,1587	-7,88284	-8,67311	-9,54243	-10,5085	-11,5957	-12,8386	-14,2896	-16,0328	-18,2165	-21,1381	-25,6751	-35,1175				
232	61	-0,28956	-0,89934	-1,55518	-2,26464	-3,03723	-3,8853	-4,82524	-5,87936	-6,45864	-7,07932	-7,7478	-8,47204	-9,26221	-10,1315	-11,0976	-12,1848	-13,4277	-14,8787	-16,6217	-18,8046	-21,7272	-26,1642	-35,7066			
231	65	-0,27146	-0,84123	-1,45101	-2,10688	-2,81631	-3,5889	-4,43697	-5,37891	-6,43103	-7,01031	-7,63099	-8,29947	-9,02371	-9,81388	-10,6832	-11,6493	-12,7364	-13,9794	-15,4304	-17,1734	-19,3563	-22,2789	-26,7158	-36,2583		
230	69	-0,25549	-0,79017	-1,35994	-1,96973	-2,62658	-3,33502	-4,10761	-4,95569	-5,89562	-6,94975	-7,52902	-8,14971	-8,81818	-9,54243	-10,3326	-11,2019	-12,168	-13,2662	-14,4981	-15,9491	-17,6921	-19,875	-22,7976	-27,2346	-36,777	
229	73	-0,24129	-0,74496	-1,27965	-1,84942	-2,4692	-3,11505	-3,8245	-4,59709	-5,44517	-6,3851	-7,43922	-8,0185	-8,63918	-9,30766	-10,0319	-10,8221	-11,6914	-12,6578	-13,7446	-14,9876	-16,4386	-18,1816	-20,3645	-23,2871	-27,724	-37,2665
228	77	-0,22859	-0,70465	-1,20832	-1,743	-2,31277	-2,92266	-3,57841	-4,28786	-5,06045	-5,90852	-6,84845	-7,90258	-8,48185	-9,10254	-9,77101	-10,4963	-11,2854	-12,1547	-13,1208	-14,208	-15,4609	-16,902	-18,645	-20,8279	-23,7604	-28,1874
227	81	-0,21716	-0,66848	-1,14453	-1,6482	-2,18289	-2,75266	-3,36245	-4,0183	-4,72774	-5,50033	-6,34841	-7,28834	-8,34247	-8,92174	-9,54243	-10,2109	-10,9351	-11,7253	-12,5946	-13,5607	-14,6479	-15,8908	-17,3418	-19,0849	-21,2677	-24,1903
226	85	-0,20682	-0,63684	-1,08715	-1,56321	-2,06688	-2,60157	-3,17134	-3,78112	-4,43687	-5,14642	-5,91901	-6,76709	-7,70702	-8,76114	-9,34042	-9,9611	-10,6296	-11,3538	-12,144	-13,0133	-13,9794	-15,0666	-16,3095	-17,7605	-19,5035	-21,6864
225	89	-0,19742	-0,60624	-1,03626	-1,48667	-1,96263	-2,4663	-3,00099	-3,57076	-4,18065	-4,8364	-5,64684	-6,31843	-7,16651	-8,10644	-9,16057	-9,73984	-10,3605	-11,029	-11,7632	-12,5434	-13,4127	-14,3788	-15,466	-16,7089	-18,1699	-19,9029
224	93	-0,18883	-0,57927	-0,9881	-1,41712	-1,86843	-2,34449	-2,84816	-3,38286	-3,95262	-4,56241	-5,21826	-5,9277	-6,70029	-7,54837	-8,4883	-9,54243	-10,1217	-10,7424	-11,4109	-12,1361	-12,9253	-13,7946	-14,7607	-15,8478	-17,0908	-18,5418
223	99	-0,35823	-0,73188	-1,12232	-1,53114	-1,96016	-2,41148	-2,88754	-3,39121	-3,92589	-4,49566	-5,10545	-5,7613	-6,47076	-7,24333	-8,09141	-9,03134	-10,0856	-10,6647	-11,2854	-11,9639	-12,6781	-13,4683	-14,3376	-15,3037	-16,3909	-17,6338
222	107	-0,33093	-1,0332	-1,40686	-1,79729	-2,20611	-2,63513	-3,08645	-3,56251	-4,06618	-4,60086	-5,17064	-5,78042	-6,43627	-7,14672	-7,91631	-8,76638	-9,70631	-10,7604	-11,3397	-11,9604	-12,6289	-13,3531	-14,1433	-15,0126	-15,9787	-17,0659
221	115	-0,3076	-0,95721	-1,65948	-2,03313	-2,42357	-2,83239	-3,26141	-3,71273	-4,18879	-4,69246	-5,22715	-5,79692	-6,4067	-7,06255	-7,772	-8,54469	-9,39266	-10,3326	-11,3867	-11,966	-12,5867	-13,2552	-13,9794	-14,7696	-15,6389	-16,605
220	123	-0,28716	-0,89164	-1,54136	-2,24363	-2,61727	-3,00772	-3,41654	-3,84556	-4,29688	-4,77294	-5,27661	-5,81129	-6,38106	-6,99086	-7,6487	-8,35615	-9,12873	-9,97681	-10,9167	-11,9709	-12,5501	-13,1708	-13,8393	-14,5635	-15,3537	-16,223
219	131	-0,26935	-0,83449	-1,43897	-2,08868	-2,79096	-3,1646	-3,55504	-3,96386	-4,39288	-4,8442	-5,32026	-5,82393	-6,35861	-6,92839	-7,53817	-8,19402	-8,90347	-9,67606	-10,5241	-11,4641	-12,5182	-13,0976	-13,7182	-14,3866	-15,1109	-15,901
218	139	-0,25362	-0,78422	-1,34936	-1,95384	-2,60355	-3,30582	-3,67947	-4,06991	-4,47873	-4,90776	-5,36907	-5,83513	-6,3388	-6,87349	-7,44326	-8,05304	-8,70889	-9,41834	-10,1909	-11,039	-11,9789	-13,0331	-13,6123	-14,233	-14,9016	-15,6257
217	147	-0,23963	-0,73967	-1,27027	-1,83541	-2,43989	-3,0896	-3,79187	-4,16552	-4,55696	-4,96478	-5,3938	-5,84512	-6,32118	-6,82465	-7,35964	-7,92931	-8,53909	-9,19494	-9,90438	-10,677	-11,5251	-12,465	-13,5191	-14,0984	-14,7191	-15,3875
216	155	-0,2271	-0,69991	-1,19996	-1,73066	-2,29569	-2,90017	-3,54989	-4,25216	-4,62581	-5,01625	-5,42507	-5,85409	-6,30541	-6,78147	-7,28514	-7,81982	-8,38959	-8,99938	-9,65523	-10,3647	-11,1373	-11,9853	-12,9253	-13,9794	-14,5587	-15,1794
215	163	-0,21581	-0,66421	-1,13703	-1,63708	-2,16768	-2,73281	-3,33729	-3,98701	-4,68928	-5,06292	-5,45337	-5,86219	-6,29121	-6,74253	-7,21889	-7,72226	-8,25684	-8,82671	-9,4366	-10,0923	-10,8018	-11,5744	-12,4225	-13,3624	-14,4165	-14,9958
214	171	-0,20569	-0,63198	-1,08038	-1,5532	-2,05325	-2,58385	-3,14898	-3,75346	-4,40318	-5,10545	-5,47909	-5,86954	-6,27836	-6,70738	-7,1587	-7,63476	-8,13843	-8,67311	-9,24288	-9,85267	-10,5086	-11,218	-11,9906	-12,8386	-13,7786	-14,8327
213	179	-0,1963	-0,60273	-1,02912	-1,47762	-1,95034	-2,45039	-2,98099	-3,54612	-4,1506	-4,80032	-5,50259	-5,87623	-6,26668	-6,6756	-7,10462	-7,55884	-8,03189	-8,53565	-9,07026	-9,64002	-10,2498	-10,9057	-11,6151	-12,3877	-13,2368	-14,1767
212	187	-0,18781	-0,57607	-0,9825	-1,40889	-1,85729	-2,33011	-2,83016	-3,36076	-3,92589	-4,53037	-5,18009	-5,88236	-6,256	-6,64645	-7,05527	-7,48429	-7,93561	-8,41187	-8,91634	-9,45002	-10,0198	-10,6296	-11,2854	-11,9949	-12,7676	-13,6165
211	195	-0,18002	-0,55167	-0,93993	-1,34636	-1,77275	-2,22116	-2,69397	-3,19402	-3,72462	-4,28975	-4,89423	-5,64396	-6,24622	-6,61986	-7,01031	-7,41913	-7,84816	-8,29947	-8,77653	-9,2792	-9,81388	-10,3837	-10,9934	-11,6493	-12,3587	-13,1313
210	203	-0,17286	-0,52925	-0,9008	-1,28916	-1,69559	-2,12188	-2,57038	-3,0432	-3,54326	-4,07365	-4,63595	-5,24348	-5,89318	-6,59545	-6,96909	-7,35964	-7,76838	-8,19738	-8,6487	-9,12475	-9,62842	-10,1631	-10,7329	-11,3427	-11,9985	-12,708
209	211	-0,16624	-0,50859	-0,86498	-1,23663	-1,62489	-2,03132	-2,45771	-2,90611	-3,37893	-3,87897	-4,40957	-4,97471	-5,57919	-6,2289	-6,93118	-7,30482	-7,69526	-8,10409	-8,53311	-8,98442	-9,46048	-9,96416	-10,4988	-11,0686	-11,6784	-12,3342
208	219	-0,16011	-0,48948	-0,83182	-1,18821	-1,55986	-1,94812	-2,35455	-2,78094	-3,22934	-3,70216	-4,20221	-4,73281	-5,29794	-5,90242	-6,55214	-7,25441	-7,62606	-8,0186	-8,42732	-8,85634	-9,30766	-9,78372	-10,2874	-10,8221	-11,3918	-12,0016
207	231	-0,30614	-0,62347	-0,95283	-1,29518	-1,65157	-2,02322	-2,41148	-2,81791	-3,2443	-3,6927	-4,16552	-4,66556	-5,19617	-5,7613	-6,36578	-7,0166	-7,71777	-8,09141	-8,48185	-8,89068	-9,3197	-9,77101	-10,2471	-10,7507	-11,2854	-11,8552
206	247	-0,28598	-0,88784	-1,20517	-1,53453	-1,87688	-2,23327	-2,60492	-2,99318	-3,39961	-3,826	-4,2744	-4,74722	-5,24726	-5,77786	-6,343	-6,94748	-7,59719	-8,29947	-8,67311	-9,06365	-9,47238	-9,9014	-10,3527	-10,8288	-11,3324	-11,8671
205	263	-0,26831	-0,83116	-1,43302	-1,75035	-2,07871	-2,42205	-2,77846	-3,15009	-3,63835	-3,94479	-4,37117	-4,81958	-5,29239	-5,79244	-6,32304	-6,88818	-7,49266	-8,14237	-8,84464	-9,21829	-9,60873	-10,0176	-10,4466	-10,8979	-11,3739	-11,8776
204	279	-0,2527	-0,78128	-1,34413	-1,94699	-2,26331	-2,59268	-2,93502	-3,29142	-3,86306	-4,05132	-4,45776	-4,88414	-5,33265	-5,80536	-6,30541	-6,83601	-7,40114	-8,00562	-8,65534	-9,35761	-9,73126	-10,1217	-10,5305	-10,9595	-11,4109	-11,8869
203	295	-0,2388	-0,73705	-1,26564	-1,82848	-2,43034	-2,74767	-3,07703	-3,41938	-3,77577	-4,14742	-4,53568	-4,94211	-5,3685	-5,8169	-6,28972	-6,78976	-7,32037	-7,8866	-8,48998	-9,1397	-9,84197	-10,2156	-10,6061	-11,0149	-11,4439	-11,8952
202	311	-0,22636	-0,69757	-1,19582	-1,7244	-2,28725	-2,88911	-3,20644	-3,5358	-3,87815	-4,23454	-4,60619	-4,99445	-5,40088	-5,82727	-6,27567	-6,74849	-7,24853	-7,77913	-8,34427	-8,94875	-9,59846	-10,3007	-10,6744	-11,0648	-11,4736	-11,9027
201	327	-0,21514	-0,6621	-1,13332	-1,63167	-2,16015	-2,723	-3,32486	-3,64219	-3,97165	-4,31389	-4,67029	-5,04193	-5,43019	-5,83663	-6,26301	-6,71142	-7,18423	-7,68428	-8,21488	-8,78002	-9,3845	-10,0342	-10,7365	-11,1101	-11,5006	-11,9094
200	343	-0,20499	-0,63007	-1,07703	-1,54824	-2,0465	-2,57608	-3,13792	-3,73979	-4,05711	-4,38648	-4,72882	-5,08522	-5,45686	-5,84612	-6,25165	-6,67794	-7,12634	-7,59916	-8,09921	-8,62981	-9,19494	-9,79942	-10,4491	-11,1514	-11,5251	-11,9155
199	359	-0,19575	-0,60099	-1,02608	-1,47304	-1,94425	-2,4425	-2,97109	-3,53393	-4,13579	-4,45312	-4,78248	-5,12483	-5,48122	-5,85287	-6,24113	-6,64756	-7,07396	-7,52236	-7,99517	-8,49521	-9,02581	-9,59096	-10,1964	-10,8461	-11,5474	-11,9211
198	375	-0,1873	-0,57448	-0,97973	-1,40481	-1,85177	-2,32298	-2,82124	-3,34982	-3,91287	-4,51453	-4,83186	-5,16122	-5,50356	-5,85996	-6,2316	-6,61986	-7,0263	-7,45268	-7,90109	-8,3739	-8,87395	-9,40455	-9,96969	-10,5742	-11,2239	-11,9262
197	391	-0,17956	-0,55021	-0,93739	-1,34264	-1,76772	-2,21468	-2,6859	-3,18416	-3,71273	-4,27558	-4,87744	-5,19477	-5,52413	-5,86647	-6,22287	-6,59461	-6,98277	-7,38921	-7,81559	-8,264	-8,73681	-9,23655	-9,76746	-10,3326	-10,9371	-11,5868
196	407	-0,17243	-0,52791	-0,89857	-1,28675	-1,69098	-2,11607	-2,56304	-3,03426	-3,5326	-4,06108	-4,62393	-5,22578	-5,54312	-5,87248	-6,21483	-6,57122	-6,94287	-7,33113	-7,73756	-8,16395	-8,61235	-9,08517	-9,58521	-10,1158	-10,6809	-11,2854
195	423	-0,16585	-0,50735	-0,86283	-1,23349	-1,62067	-2,02591	-2,45099	-2,89796	-3,36917	-3,85742	-4,396	-4,95565	-5,56071	-5,87804	-6,2074	-6,54976	-6,90614	-7,27779	-7,66605	-8,07248	-8,49886	-8,94727	-9,42009	-9,92013	-10,4507	-11,0159
194	439	-0,15975	-0,48833	-0,82983	-1,18531	-1,55597	-1,94315	-2,34839	-2,77348	-3,22044	-3,69165	-4,1899	-4,71549	-5,28133	-5,88319	-6,20062	-6,52988	-6,87223	-7,22862	-7,60027	-7,98853	-8,39496	-8,82135	-9,26975	-9,74257	-10,2426	-10,7732
193	455	-0,15408	-0,47068	-0,79927	-1,14077	-1,49625	-1,86691	-2,26409	-2,65933	-3,08441	-3,53138	-4,00259	-4,50084	-5,02942	-5,59227	-6,19413	-6,51146	-6,84082	-7,18317	-7,53966	-7,91121	-8,29947	-8,7059	-9,13229	-9,58069	-10,0535	-10,5536
192	471	-0,1488	-0,45427	-0,77087	-1,09946	-1,44096	-1,79644	-2,1671	-2,55428	-2,95952	-3,3846	-3,83157	-4,30278	-4,80103	-5,32961	-5,89246	-6,49432	-6,81166	-7,14101	-7,48336	-7,83975	-8,2114	-8,59966	-9,00609	-9,43248	-9,85088	-10,3537

191	495	-0,28539	-0,55048	-0,88595	-1,20256	-1,53114	-1,87265	-2,22813	-2,59878	-2,98596	-3,39121	-3,81629	-4,26325	-4,73447	-5,23272	-5,7613	-6,32416	-6,92601	-7,24333	-7,5727	-7,91504	-8,27144	-8,64308	-9,03134	-9,43777	-9,86416	-10,3126
190	527	-0,26779	-0,5295	-1,12459	-1,43006	-1,74667	-2,07525	-2,41675	-2,77224	-3,14289	-3,53007	-3,93532	-4,3604	-4,80736	-5,27857	-5,77683	-6,30541	-6,86825	-7,47012	-7,78744	-8,11661	-8,45915	-8,81554	-9,18719	-9,57645	-9,98188	-10,4083
189	559	-0,25224	-0,77982	1,34153	-1,63662	1,94209	-2,25869	-2,58727	-2,92878	-3,28426	-3,65491	-4,04209	-4,44734	-4,87242	5,31938	-5,7906	-6,28885	-6,81743	-7,38028	-7,98214	-8,29947	-8,62883	8,97117	-9,32757	-9,69921	10,0875	10,4939
188	591	-0,23839	-0,73575	-1,26333	-1,82504	-2,12013	-2,4256	-2,7422	-3,07079	-3,41229	-3,76777	-4,13843	-4,52561	-4,93085	-5,35594	-5,8029	-6,27411	-6,77236	-7,30095	-7,86379	-8,46565	8,78298	9,11234	-9,45469	-9,81108	-10,1827	-10,671
187	623	-0,22589	-0,6964	-1,19376	-1,72134	-2,28305	-2,57814	-2,88361	-3,20022	-3,5288	-3,8703	-4,22579	-4,59644	-4,98362	-5,38886	-5,81395	-6,26091	-6,73212	-7,23038	-7,75896	-8,3218	-8,92366	9,24099	-9,57035	-9,9127	10,2691	-10,6407
186	655	-0,21481	-0,66105	-1,13147	-1,62883	-2,15641	-2,71812	-3,01321	-3,31868	-3,63528	-3,96386	-4,30537	-4,66085	-5,0315	-5,41868	-5,82393	-6,24901	-6,69597	-7,16719	-7,66544	-8,19402	-8,75687	9,35873	-9,67606	-10,0054	-10,3478	-10,7042
185	687	-0,20468	-0,62912	-1,07536	-1,54578	-2,04314	-2,57072	-3,13242	-3,42751	-3,73298	-4,04959	-4,37817	-4,71968	-5,07516	-5,44581	-5,83299	-6,23824	-6,66332	-7,11028	-7,5815	-8,07975	8,60833	-9,17118	-9,77304	-10,0904	-10,4197	-10,7621
184	719	-0,19547	-0,60013	-1,02456	-1,4708	-1,94122	-2,43858	-2,96616	-3,52787	-3,82296	-4,12843	-4,44503	-4,77362	-5,11512	-5,4706	-5,84126	-6,22844	-6,63368	-7,05876	-7,50573	-7,97694	8,47519	-9,00377	-9,56662	-10,1685	-10,4858	-10,8152
183	751	-0,18705	-0,57369	-0,97835	-1,40275	-1,84902	-2,31944	-2,8168	-3,34438	-3,90609	-4,20118	-4,50665	-4,82325	-5,15184	-5,49334	-5,84882	-6,21948	-6,60666	-7,0119	-7,43699	-7,88395	8,35516	8,86341	-9,382	-9,94484	-10,5467	-10,864
182	783	-0,17933	-0,54949	-0,93613	-1,34078	-1,76522	-2,21146	-2,68188	-3,17924	-3,70682	-4,26852	-4,56362	-4,86908	-5,18569	-5,51427	-5,85578	-6,21126	-6,58191	-6,96909	-7,37434	-7,79942	8,24638	8,7176	-9,21585	-9,74443	-10,3073	-10,9091
181	815	-0,17222	-0,52724	-0,89741	-1,28405	-1,6887	-2,11314	-2,55938	-3,0298	-3,52716	-4,05473	-4,61644	-4,91163	-5,217	-5,53361	-5,86219	-6,20369	-6,55918	-6,92983	-7,31701	-7,72226	-8,14734	-8,5943	-9,06561	-9,56377	-10,0923	-10,6552
180	847	-0,16565	-0,50673	-0,86178	-1,23192	-1,61656	-2,02322	-2,44765	-2,89389	-3,36431	-3,86167	-4,38925	-4,95096	-5,24605	-5,55152	-5,86812	-6,19671	-6,53821	-6,89369	-7,26435	-7,65153	-8,05677	8,48185	-8,92882	-9,40003	-9,89828	-10,4269
179	879	-0,15956	-0,48776	-0,82884	-1,18387	-1,55403	-1,94067	-2,34533	-2,76976	-3,216	-3,68642	-4,18378	-4,71136	-5,27307	-5,56816	-5,87363	-6,19023	-6,51882	-6,86032	-7,2158	-7,58646	-7,97364	-8,37888	-8,80396	-9,25092	-9,72214	-10,2204
178	911	-0,15391	-0,47015	-0,79835	-1,13943	-1,49445	-1,86462	-2,25126	-2,65592	-3,08035	-3,52659	-3,99701	-4,49437	-5,02195	-5,58366	-5,87875	-6,18422	-6,50082	-6,82941	-7,17091	-7,52638	-7,89705	-8,28423	-8,68947	-9,11455	-9,56151	-10,0327
177	943	-0,14864	-0,45377	-0,77002	-1,09821	-1,4393	-1,79433	-2,16449	-2,55113	-2,95578	-3,38022	-3,82646	-4,29688	-4,79424	-5,32182	-5,88352	-6,17861	-6,48408	-6,80069	-7,12927	-7,47078	7,82626	-8,19691	-8,58409	-8,98934	-9,41442	-9,86138
176	975	-0,14372	-0,4385	-0,74363	-1,05988	1,38807	-1,72916	-2,08419	-2,45435	2,84099	-3,24564	-3,67008	-4,11632	-4,58674	-5,0841	-5,61167	-6,17338	-6,46847	-6,77394	-7,09055	-7,41913	-7,76063	8,11612	-8,48677	-8,87395	-9,2792	-9,70428
175	1023	-0,27604	-0,56114	-0,85592	-1,16105	-1,4773	-1,80649	-2,14658	-2,50161	-2,87177	-3,25841	-3,66306	-4,0875	-4,53374	-5,00416	-5,50152	-6,02909	-6,5908	-6,88589	-7,19136	-7,50797	-7,83655	-8,17805	-8,53354	-8,90419	-9,29137	-9,69662
174	1087	-0,25964	-0,80312	-1,08822	-1,383	-1,68813	-2,00437	-2,33257	-2,67366	-3,02868	-3,39884	-3,78548	-4,19014	-4,61457	-5,06082	-5,53123	-6,02889	-6,55617	-7,11788	-7,41297	-7,71844	-8,03505	-8,36363	-8,70513	-9,06062	-9,43127	9,81846
173	1151	-0,2449	-0,75646	-1,30003	-1,58513	-1,87991	-2,18505	-2,50129	-2,82949	-3,17057	-3,5256	-3,89576	-4,2824	-4,68706	-5,11149	-5,55773	-6,02815	-6,52551	-7,05309	-7,6148	-7,90989	-8,21536	-8,53196	-8,86064	-9,20205	-9,55753	-9,92819
172	1215	-0,23183	-0,71492	-1,22648	-1,77005	2,05615	-2,34993	-2,65506	-2,97131	3,29951	-3,64059	-3,99562	-4,36578	-4,78242	-5,15708	-5,58161	-6,02776	-6,49817	-6,99553	-7,52311	-8,08482	-8,37991	-8,68538	-9,00198	-9,33066	-9,67207	-10,0275
171	1279	-0,22008	-0,67772	-1,16081	-1,67236	-2,21594	2,50104	-2,79582	-3,10095	-3,41719	-3,74639	-4,08648	-4,4415	-4,81166	-5,1983	-5,60298	-6,02739	-6,47364	-6,94405	-7,44141	-7,96899	-8,5307	-8,82579	-9,13126	-9,44787	-9,77645	-10,116
170	1343	-0,20947	-0,64419	-1,10183	-1,58492	-2,09647	-2,64005	-2,92515	-3,21993	-3,52506	-3,8413	-4,1695	-4,51058	-4,86561	-5,23577	-5,62241	-6,02707	-6,4515	-6,89775	-7,36816	-7,86552	-8,3931	8,95481	-9,2499	-9,55537	-9,87197	-10,2006
169	1407	-0,19983	-0,61383	-1,04855	-1,50619	-1,98928	-2,50083	-3,04441	-3,32951	-3,62429	-3,92942	-4,24567	-4,57386	-4,91495	-5,26997	-5,64014	-6,02678	-6,43143	-6,85586	-7,30211	-7,77253	-8,26989	-8,79746	-9,35917	-9,65426	-9,95973	-10,2763
168	1471	-0,19104	-0,5862	-1,0002	-1,43492	-1,89256	-2,37565	-2,8872	-3,43078	-3,71588	-4,01066	-4,31579	-4,63204	-4,96023	-5,30132	-5,65635	-6,02651	-6,41315	-6,8178	-7,24224	-7,68848	-8,1589	-8,65626	-9,18384	-9,74554	-10,0406	-10,3461
167	1535	-0,18299	-0,56095	-0,95611	-1,37011	-1,80484	-2,26247	-2,74557	-3,25712	-3,80069	-4,0858	-4,38057	-4,68571	-5,00195	-5,33015	-5,67123	-6,02626	-6,39642	-6,78306	-7,18772	-7,61215	-8,05839	-8,52881	-9,02617	-9,55375	-10,1155	-10,4105
166	1599	-0,17559	-0,53779	-0,91575	-1,31092	-1,72492	-2,15964	-2,61727	-3,10037	-3,61192	-4,1555	-4,4406	-4,73537	-5,04051	-5,35675	-5,68495	-6,02603	-6,38106	-6,76122	-7,13786	-7,54252	-7,96695	8,4132	-8,88361	-9,38097	-9,90855	10,4703
165	1663	-0,16877	-0,51647	-0,87867	-1,25663	-1,65179	-2,06579	-2,50052	-2,95815	-3,44124	-3,9528	-4,49637	-4,78147	-5,07625	-5,38138	-5,69763	-6,02582	-6,36691	-6,72194	-7,0921	-7,47874	-7,88339	-8,30783	-8,75407	-9,22449	-9,72185	-10,2494
164	1727	-0,16245	-0,49677	-0,84447	-1,20667	-1,58463	-1,97979	-2,39379	-2,82852	-3,28615	-3,76925	-4,2808	-4,82437	-5,10947	-5,40425	-5,70939	-6,02563	-6,35383	-6,69491	-7,04994	-7,4201	-7,80674	-8,2114	-8,63583	-9,08207	-9,55249	-10,0499
163	1791	-0,15659	-0,47852	-0,81283	-1,16053	-1,52273	-1,9007	-2,29586	-2,70986	-3,14458	-3,60222	-4,08531	-4,59686	-5,14044	-5,42554	-5,72032	-6,02545	-6,3417	-6,66989	-7,01098	-7,366	-7,73616	-8,12281	-8,52746	-8,95189	-9,39814	-9,86855
162	1855	-0,15114	-0,46166	-0,78348	-1,1178	1,4655	-1,8277	-2,20566	-2,60082	3,01482	-3,44955	-3,90718	-4,39028	-4,90183	-5,44541	-5,73051	-6,02528	-6,33042	-6,64666	-6,97486	-7,31594	-7,67097	-8,04113	-8,42777	-8,83243	-9,25686	-9,7031
161	1919	-0,14606	-0,44577	-0,75618	-1,07811	-1,41242	-1,76012	-2,12232	-2,50028	-2,89545	-3,30945	-3,74417	-4,2018	-4,6849	-5,19645	-5,74003	-6,02513	-6,3199	-6,62504	-6,94128	-7,26948	7,61056	-7,96559	-8,33575	-8,72239	9,12705	-9,55148
160	1983	-0,14131	-0,43102	-0,73072	-1,04114	-1,36306	-1,69738	-2,04507	-2,40727	-2,78524	-3,1804	-3,5944	-4,02913	-4,48678	-4,96985	-5,48141	-6,02498	-6,31008	-6,60486	-6,90999	-7,22624	-7,55443	-7,89552	-8,25055	-8,62071	-9,00735	-9,412
159	2079	-0,27159	-0,55194	-0,84165	-1,14136	-1,45177	-1,7737	-2,10801	-2,45571	-2,81791	-3,19587	-3,59104	-4,00504	-4,43976	-4,89739	-5,38049	-5,89204	-6,43562	-6,72072	-7,0155	-7,32063	-7,63687	-7,96507	-8,30615	-8,66118	-9,03134	-9,41798
158	2207	-0,2556	-0,79055	-1,0709	-1,36061	-1,66031	-1,97073	-2,29265	-2,62697	2,97467	-3,33687	-3,71483	-4,10999	-4,52399	-4,95872	-5,41635	-5,89944	-6,411	-6,95457	-7,23967	-7,53445	-7,83959	-8,15583	-8,48403	-8,82511	-9,18014	-9,5503
157	2335	-0,2414	-0,74629	-1,28024	-1,56059	-1,8503	-2,15	-2,46042	-2,78234	-3,11666	-3,48436	-3,82656	-4,20452	-4,59968	-5,01368	-5,44841	-5,90604	-6,38914	-6,90069	-7,44426	-7,72937	-8,02414	-8,32928	-8,64552	-8,97372	-9,3148	-9,66983
156	2463	-0,22868	-0,70495	-1,20884	-1,74379	-2,02414	-2,31385	-2,61355	-2,92397	-3,24589	-3,58021	-3,92791	-4,29011	-4,66807	-5,06323	-5,47723	-5,91196	-6,36959	-6,85259	-7,36424	-7,90782	-8,19292	-8,48769	-8,79283	-9,10907	-9,43727	-9,77836
155	2591	-0,21724	-0,66874	-1,14601	-1,6489	-2,18385	-2,4642	-2,75391	-3,05361	-3,36403	-3,68595	-4,02027	-4,36797	-4,73017	-5,10813	-5,50329	-5,91729	-6,35202	-6,80965	-7,29275	-7,8043	-8,34788	-8,63298	-8,92775	-9,23289	-9,54913	-9,87733
154	2719	-0,20689	-0,63608	-1,08758	-1,56384	-2,06774	-2,60258	-2,88304	-3,17275	-3,47245	-3,78287	-4,10479	-4,4381	-4,7868	-5,149	-5,52697	-5,92213	-6,33613	-6,77086	-7,22849	-7,71158	-8,22313	-8,76671	-9,05181	-9,34659	-9,65172	-9,96797
153	2847	-0,19749	-0,60646	-1,03564	-1,48714	-1,96341	-2,46731	-3,00225	-3,2826	-3,57231	-3,87202	-4,18243	-4,50436	-4,83867	-5,18637	-5,54867	-5,92653	-6,3217	-6,7367	-7,17042	-7,62805	-8,11115	-8,6227	-9,14628	-9,45138	-9,74615	-10,0513
152	2975	-0,1889	-0,57948	-0,98846	-1,41763	-1,86913	-2,3454	-2,8493	-3,38424	-3,66459	-3,9543	-4,25401	-4,56442	-4,88635	-5,22066	-5,56836	-5,93056	-6,30852	-6,70369	-7,11769	-7,55241	-8,01004	-8,49314	-9,00469	-9,54827	-9,83337	-10,1281
151	3103	-0,18102	-0,55478	-0,94537	-1,35435	-1,78353	-2,23503	-2,71129	-3,21519	-3,75013	-4,03049	-4,3202	-4,6199	-4,93032	-5,25224	-5,58656	-5,93426	-6,29646	-6,67442	-7,06958	-7,48358	-7,91831	-8,37594	-8,85903	-9,37069	-9,91416	-10,1993
150	3231	-0,17378	-0,53213	-0,9059	-1,29848	-1,70545	-2,13463	-2,58613	-3,0624	-3,5663	-4,10124	-4,38159	-4,6713	-4,97101	-5,28142	-5,60333	-5,93766	-6,28536	-6,64756	-7,02552	-7,42069	-7,83469	-8,26941	-8,72704	-9,21014	-9,72169	-10,2663
149	3359	-0,16709	-0,51124	-0,86959	-1,24336	-1,63394	-2,04291	-2,4721	-2,9236	-3,39986	-3,90376	-4,4387	-4,71905	-5,00876	-5,30847	-5,61888	-5,94081	-6,27512	-6,62282	-6,98502	-7,36298	-7,75815	-8,17216	-8,60687	-9,06451	-9,5476	-10,0592
148	3487	-0,1609	-0,49193	-0,83608	-1,19442	-1,56819	-1,95877	-2,36775	-2,79693	-3,24843	-3,7247	-4,2286	-4,76354	-5,04389	-5,3336	-5,63331	-5,94372	-6,26564	-6,59996	-6,94766	-7,30986	-7,68782	-8,08298	-8,49699	-8,93171	-9,38934	-9,87244
147	3615	-0,15515	-0,47403	-0,80506	-1,1492	-1,50755	-1,88132	-2,2719	-2,68088	-3,11006	-3,56156	-4,03782	-4,54172	-5,07667	-5,35702	-5,64673	-5,94643	-6,25685	-6,57877	-6,91309	-7,26079	-7,62299	-8,00095	-8,39611	-8,81011	-9,24484	-9,70247
146	3743	-0,1498	-0,45738	-0,77626	-1,10729	-1,45144	-1,80978	-2,18355	-2,57413	-2,98311	-3,41229	-3,86379	-4,34006	-4,84396	-5,3789	-5,65925	-5,94896	-6,24866	-6,55908	-6,881	-7,21532	-7,56302	-7,92522	-8,30318	-8,69834	-9,11234	-9,54707
145	3871	-0,14481	-0,44187	-0,74945	-1,06833	-1,39936	-1,7435	-2,10185	-2,47562	-2,8662	-3,27517	-3,70436	-4,15586	-4,63212	-5,13602	-5,67096	-5,95132	-6,24103	-6,54073	-6,85115	-7,17307	-7,50738	-7,85508	-8,21728	-8,59526	-8,99041	-9,40441
144	3999	-0,14013	-0,42737	-0,72443	-1,03201	-1,35089	-1,68192	-2,02607	-2,38441	-2,75818	-3,14876	-3,55774	-3,98692	-4,43842	-4,91469	-5,41858	-5,95353	-6,23388	-6,52359	-6,82329	-7,13371	-7,45563	-7,78995	-8,13765	-8,49985	-8,87781	-9,27297
143	4191	-0,26942	-0,64746	-0,8347	-1,13176	-1,43934	-1,75822	-2,08924	-2,43339	-2,79174	-3,16551	-3,55609	-3,96506	-4,39426	-4,84576	-5,32201	-5,82591	-6,36085	-6,64121	-6,93092	-7,23062	-7,54104	-7,86296	-8,19727	-8,54497	-8,90717	-9,28514
142	4447	-0,25368	-0,78441	-1,06246	-1,34969	-1,64675	-1,96433	-2,2732	-2,60423	-2,94838	-3,30673	-3,6806	-4,07108	-4,48005	-4,90924	-5,36074	-5,837	-6,3409	-6,87584	-7,1562	-7,4459	-7,74561	-8,05603	-8,37795	-8,71226	-9,05996	-9,42216
141	4703	-0,23988	-0,73984	-1,27058	-1,5488	-1,83684	-2,1329	-2,44049	-2,75936	-3,09039	-3,43454	-3,79289	-4,16686	-4,56724	-4,96621	-5,39539	-5,84689	-6,32316	-6,82706	-7,362	-7,64235	-7,93206	-8,23177	-8,54218	-8,86411	-9,19842	-9,54612
140	4959	-0,22714	-0,70008	-1,20022	-1,73095	-2,00889	-2,29622	-2,59329	-2,90087	-3,21974	-3,55077	-3,89492	-4,25327	-4,62704	-5,01762	-5,42659	-5,85578	-6,30728	-6,78354	-7,28744	-7,82238	-8,10274	-8,39244	-8,69215	-9,00257	-9,32449	-9,6588
139	5215	-0,21585	-0,66435	-1,13726	-1,63742	-2,16815	-2,44619	-2,73343	-3,03049	-3,33807	-3,65695	-3,98798	-4,33212	-4,69047	-5,06424	-5,45482	-5,8638	-6,29298	-6,74448	-7,22074	-7,72464	-8,25959	-8,53994	-8,82965	-9,12935	-9,43977	-9,76169
138	5471	-0,20563	-0,6321	-1,08058	-1,55351	-2,05367	-2,5844	-2,86244	-3,14968	-3,44674	-3,75432	-4,0732	-4,40423	-4,74837	-5,10672	-5,48049	-5,87107	-6,28004	-6,70923	-7,16073	-7,63699	-8,14089	-8,67583	-8,95619	-9,2459	-9,5456	-9,85602
137	5727	-0,19633	-0,60254	-1,02931	-1,4778	-1,95072	-2,45088	-2,98161	-3,25965	-3,54689	-3,84395	-4,15153	-4,47041	-4,80144	-5,14568	-5,50393	-5,8777	-6,26828	-6,67725	-7,10644	-7,55794	-8,0342	-8,5381	-9,07304	-9,3534	-9,64311	-9,94281
136	5983	-0,18784	-0,57617	-0,98268	-1,40916	-1,85764	-2,33056	-2,83072	-3,36144	-3,63949	-3,92672	-4,22378	-4,53137	-4,85024	-5,18127	-5,52542	-5,88377	-6,25754	-6,64812	-7,05709	-7,48628	-7,93778	-8,41404	-8,91794	-9,45288	-9,73323	-10,0229
135	6239	-0,18006	-0,55176	-0,94009	-1,3466	-1,77306	-2,22156	-2,69448	-3,19464	-3,72536	-4,0034	-4,29064	-4,5877	-4,89528	-5,21416	-5,54519	-5,88934	-6,24769	-6,62146	-7,01204	-7,42101	-7,85019	-8,30169	-8,77796	-9,28186	-9,8168	-10,0972
134	6495	-0,17289	-0,52934	-0,90108	-1,28937	-1,69588	-2,12238	-2,57084	-3,04376	-3,54392	-4,07465	-4,35269	-4,63993	-4,93699	-5,24457	-5,56346	-5,89447	-6,23862	-6,59697	-6,97074	-7,36132	-7,77029	-8,19948	-8,65098	-9,12724	-9,63114	-10,1661
133	6751	-0,16627	-0,50867	-0,86512	-1,23682	1,62515	-2,03166	-2,45813	2,90662	3,37954	-3,8797	-4,41043	4,68847	-4,9757	-5,27277	5,58035	-5,89922	-6,23025	-6,5744	6,93275	-7,30662	-7,6971	-8,10607	-8,53526	-8,98676	-9,46302	9,96692
132	7007	-0,16014	-0,48965	-0,83196	-1,1884	-1,5601	-1,94843	-2,35494	-2,78141	-3,2299	-3,70282	-4,20298	-4,73371	-5,01175	-5,29898	-5,59605	-5,90363	-6,2225	-6,55353	-6,89768	-7,25603	-7,6298	-8,02038	-8,42935	-8,85854	-9,31004	-9,7863
131	7263	-0,15444	-0,47181	-0,80123	-1,14362	-1,50008	-1,87178	-2,26011	-2,66662	-3,09309	-3,54158	-4,0145	-4,51466	-5,04538	-5,32343	-5,61066	-5,90773	-6,21531	-6,53418	-6,86521	-7,20936	-7,56771	7,94148	-8,33206	-8,74103	-9,17022	-9,62172
130	7519	-0,14914	-0,45532	-0,7727	-1,10211	-1,4445	-1,80096	-2,17266	-2,56099	-2,9675	-3,39397	-3,84246	-4,31538	-4,81554	-5,34627	-5,62431	-5,91154	-6,20861	-6,51619	-6,83508	-7,16609	-7,51024	-7,86859	-8,24236	-8,63294	-9,04191	-9,4711
129	7775	-0,14419	-0,43994	-0,74613	-1,0635	-1,39291	-1,73531	-2,09176	-2,46347	-2,8618	-3,2583	-3,68477	-4,13327	-4,60619	-5,10634	-5,63707	-5,91511	-6,20235	-6,49941	-6,80698	-7,12587	-7,4569	-7,80105	-8,15939	-8,53316	-8,92374	-9,33272
128	8031	-0,13965	-0,42667	-0,72133	-1,02751	-1,34489	-1,6743	-2,0167	-2,37315	-2,74485	-3,13318	-3,53969	-3,96615	-4,41466	-4,88757	-5,38773	-5,91846	-6,1965	-6,48373	-6,7808	-7,08838	-7,40725	-7,73826	-8,08243	-8,44076	-8,81455	-9,20513

3.0 dB 4.5 dB 6.0 dB 7.5 dB

A2

```
/* ******************************************************************** */
/*
 *      pcmmap.c
 *
 *      (C) 1997 VoCAL Technologies Ltd.
 *
 *      ALL RIGHTS RESERVED.  PROPRIETARY AND CONFIDENTIAL.
 *
 *      VoCAL Technologies Ltd.
 *      3032 Scott Blvd.
 *      Santa Clara, CA 95054
 *
 *      Product         C
 *
 *      Module:                 PCM
 *
 *      This file contains the PCM mapping functions.
 *
 *      Revision Number:        $Revision$
 *      Revision Status:        $State$
 *      Last Modified:          $Date$
 *      Identification:         $Id$
 *
 *      Revision History:       $Log$
 *      Revision 1.0  1997/03/01  00:00:00  VD
 *      Initial release of software
 *
 */
/* ******************************************************************** */

#include "standard.h"
#include "pcm.h"
#include <stdio.h>
#include <math.h>

/* ******************************************************************** */

struct law_array_s {
        sint15 code;
        sint15 linear;
        sint15 other;
        sint15 deselect;
};

struct law_s {
        float tx_power_db;
        float rx_power_db;
        sint15 count;
        sint15 dmin;
        sint15 indistinguishable;
        sint15 x;
        sint31 bit_rate;
        struct law_array_s value[128];
} u_law, u_law_rx;

#define LAW_SELECTED                            0
#define LAW_DESELECTED_MIN                      0x0001
#define LAW_DESELECTED_MAX                      0x0002
#define LAW_DESELECTED_DMIN                     0x0004
#define LAW_DESELECTED_AVOID            0x0008
#define LAW_DESELECTED_DUPLICATE        0x0010
#define LAW_DESELECTED_POWER            0x0020

/* ******************************************************************** */

#define MAP_FRAME_SIZE                          6

uint48 frame_slot_modulus[MAP_FRAME_SIZE];
uint48 total_symbols_per_frame;

/* ******************************************************************** */

#define USE_4D_TRELLIS

/* ******************************************************************** */

void
u_law_init (struct law_s *law)
{
```

B1

```
        sint15 idx, pcm;

        law->tx_power_db = 0.0;
        law->rx_power_db = 0.0;
        law->count = 0;
        law->indistinguishable = 0;
        law->dmin = 0;
        law->bit_rate = 0L;

        for (idx = 0; idx < 128; ++idx) {
                pcm = 255 - idx;

                law->value[idx].code = pcm;
                law->value[idx].linear = u_law_pcm_decode (pcm);
                law->value[idx].other = u_law_pcm_threshold (pcm);
                law->value[idx].deselect = LAW_SELECTED;
        }
}
/* ******************************************************************** */

#define SELECT_LARGER           /* appears to work better with standard 3 and 6 dB attenuation */

void
u_law_attenuate (struct law_s *law, float attn_db)
{
        sint15 idx, pcm;
        float attn;

        attn = db_to_float (-attn_db);

        law->tx_power_db = 0.0;
        law->rx_power_db = 0.0;
        law->count = 0;
        law->indistinguishable = 0;
        law->dmin = 0;
        law->bit_rate = 0L;

        for (idx = 0; idx < 128; ++idx) {
                pcm = 255 - idx;

                law->value[idx].other = (sint15) ((float) (u_law_pcm_decode (pcm)) * attn);
                law->value[idx].code = u_law_pcm_encode (law->value[idx].other);
                law->value[idx].linear = u_law_pcm_decode (law->value[idx].code);
                law->value[idx].deselect = LAW_SELECTED;
//              if (idx != 0) {
//                      if (law->value[idx].code == law->value[idx-1].code) {
//#ifdef SELECT_LARGER
//                              law->value[idx-1].deselect = LAW_DESELECTED_DUPLICATE;
//#else
//                              law->value[idx].deselect = LAW_DESELECTED_DUPLICATE;
//#endif
//                      }
//              }
        }
}
/* ******************************************************************** */

void
u_law_range (struct law_s *law, sint15 min, sint15 max)
{
        sint15 idx;

        for (idx = 0; idx < 128; ++idx) {
                if (law->value[idx].linear < min) {
                        law->value[idx].deselect |= LAW_DESELECTED_MIN;
                }

                if (law->value[idx].linear > max) {
                        law->value[idx].deselect |= LAW_DESELECTED_MAX;
                }
        }
}
/* ******************************************************************** */

void
u_law_dmin (struct law_s *law, sint15 dmin)
```

B2

```
{
        sint15 idx, pcm;
        sint15 current;

        current = law->value[125].linear;
        law->value[127].deselect |= LAW_DESELECTED_AVOID;
        law->value[126].deselect |= LAW_DESELECTED_AVOID;

        for (idx = 124; idx >= 0; --idx) {
                if (law->value[idx].linear > (current - dmin)) {
                        law->value[idx].deselect |= LAW_DESELECTED_DMIN;
                }
                else {
                        current = law->value[idx].linear;
                }
        }
}
/* ************************************************************************ */

void
u_law_discard_indinstinguishable (struct law_s *law)
{
        sint15 idx;

        for (idx = 1; idx < 125; ++idx) {
                if (law->value[idx].code == law->value[idx-1].code) {
//#ifdef SELECT_LARGER
//                      law->value[idx-1].deselect |= LAW_DESELECTED_DUPLICATE;
//#else
//                      law->value[idx].deselect |= LAW_DESELECTED_DUPLICATE;
//#endif
                        if (law->value[idx-1].deselect & (LAW_DESELECTED_DMIN |
LAW_DESELECTED_AVOID | LAW_DESELECTED_POWER)) {
                                law->value[idx-1].deselect |= LAW_DESELECTED_DUPLICATE;
                        }
                        else {
                                law->value[idx].deselect |= LAW_DESELECTED_DUPLICATE;
                        }
                }
        }
}
/* ************************************************************************ */

void
u_law_attenuate_dmin (struct law_s *tx_law, struct law_s *rx_law)
{
        sint15 idx;

        rx_law->value[127].deselect |= LAW_DESELECTED_AVOID;
        rx_law->value[126].deselect |= LAW_DESELECTED_AVOID;

        for (idx = 124; idx >= 0; --idx) {
                if (tx_law->value[idx].deselect & LAW_DESELECTED_DMIN) {
                        rx_law->value[idx].deselect |= LAW_DESELECTED_DMIN;
                }
        }
}
/* ************************************************************************ */

void
u_law_exclude (struct law_s *law)
{
        sint15 idx, pcm;

        for (idx = 127; idx >= 0; --idx) {
                pcm = 255 - idx;

                if ((pcm == 165) || (pcm == 169)) {
                        law->value[idx].deselect |= LAW_DESELECTED_AVOID;
                }
        }
}
/* ************************************************************************ */

float
u_law_power (struct law_s *law, float max_db)
```

B3

```
{
        sint15 idx, count, dmin, prev;
        float factor, power, max_float;
        float tx_power, tx_sum, rx_power, rx_sum;

        factor = pow (10.0, ((3.17 + 3.01244) / 20.0)) / 8159.0;

        count = 0;
        tx_sum = 0.0;
        rx_sum = 0.0;
        dmin = 8159;
        prev = -8159;

        max_float = db_to_power (max_db);
        for (idx = 0; idx < 128; ++idx) {
                if (law->value[idx].deselect == 0) {
                        rx_power = law->value[idx].linear * factor;
                        rx_power = rx_power * rx_power;
                        rx_power = rx_power + rx_sum;

                        tx_power = u_law.value[idx].linear * factor;
                        tx_power = tx_power * tx_power;
                        tx_power = tx_power + tx_sum;

                        power = tx_power / (float) (count + 1);
                        if (power <= max_float) {
                                rx_sum = rx_power;
                                tx_sum = tx_power;
                                count++;
                                if ((law->value[idx].linear - prev) < dmin) {
                                        dmin = law->value[idx].linear - prev;
                                }
                                prev = law->value[idx].linear;
                        }
                        else {
                                law->value[idx].deselect = LAW_DESELECTED_POWER;
                        }
                }
//printf ("%f  %f  %d  %d\n", power, tx_sum, count, law->value[idx].deselect);
        }
        tx_power = tx_sum / (float) count;
        tx_power = power_to_db (tx_power);

        rx_power = rx_sum / (float) count;
        rx_power = power_to_db (rx_power);

        law->tx_power_db = tx_power;
        law->rx_power_db = rx_power;
        law->count = count;
        law->dmin = dmin;

        return power;
}
/* ************************************************************************** */

void
u_law_count_indinstinguishable (struct law_s *law)
{
        sint15 idx, count;

        count = 0;
        for (idx = 1; idx < 128; ++idx) {
                if (law->value[idx].deselect == LAW_DESELECTED_DUPLICATE) {
                        count++;
                }
        }
        law->indistinguishable = count;
}
/* ************************************************************************** */

sint31
u_law_bit_rate (struct law_s *law)
{
        float bit_rate_float;

        bit_rate_float = (log10 ((float) law->count * 2.0) / log10(2.0)) * 8000.0;
        law->bit_rate = (sint31) bit_rate_float;
```

B4

```
        return law->bit_rate;
}

/* ****************************************************************** */

//#define MAX_SHIFT 20            /* covers to -7.08 dB */
#define MAX_SHIFT 30          /* covers to -11.2 dB */
//#define MAX_SHIFT 40            /* covers to -14 65 dB */

float
u_law_detect_attenuation (struct law_s *law)
{
        sint15 idx, idx1, idx2, idx3, trim;
        sint15 count;
        float attn_low[MAX_SHIFT], attn_high[MAX_SHIFT];
        float a_low[MAX_SHIFT], a_high[MAX_SHIFT];
        float attn, diff;

        count = 0;
        idx3 = 0;

        for (idx = 128; idx >= MAX_SHIFT; --idx) {
                if (law->value[idx].deselect & LAW_DESELECTED_DUPLICATE) {
                        idx3++;
                        if (count == 0) {
                                for (count = 0; count < MAX_SHIFT; ++count) {
#ifdef SELECT_LARGER
                                        attn_high[count] = (float) u_law.value[idx-count].other / (float) u_law.value[idx+1].linear;
                                        attn_low[count] = (float) u_law.value[idx-count-1].other / (float) u_law.value[idx].linear;
#else
                                        attn_high[count] = (float) u_law.value[idx-count-1].other / (float) u_law.value[idx].linear;
                                        attn_low[count] = (float) u_law.value[idx-count-2].other / (float) u_law.value[idx-1].linear;
#endif
//                                      printf ("original %d %f %f\n", u_law.value[idx].code, attn_high[count], attn_low[count]);
                                }
                                count = MAX_SHIFT;
                        }
                        else {
                                for (idx1 = 0; idx1 < MAX_SHIFT; ++idx1) {
#ifdef SELECT_LARGER
                                        a_high[idx1] = (float) u_law.value[idx-idx1].other / (float) u_law.value[idx+1].linear;
                                        a_low[idx1] = (float) u_law.value[idx-idx1-1].other / (float) u_law.value[idx].linear;
#else
                                        a_high[idx1] = (float) u_law.value[idx-idx1-1].other / (float) u_law.value[idx].linear;
                                        a_low[idx1] = (float) u_law.value[idx-idx1-2].other / (float) u_law.value[idx-1].linear;
#endif
//                                      printf ("generating %d %f %f\n", u_law.value[idx].code, a_high[idx1], a_low[idx1]);
                                }

                                for (idx1 = 0; idx1 < count; ++idx1) {
                                        trim = 1;
                                        for (idx2 = 0; idx2 < MAX_SHIFT; ++idx2) {
                                                if ((attn_high[idx1] >= a_high[idx2]) && (attn_low[idx1] <= a_high[idx2])) {
//                                                      printf ("reduce high %f %f %f\n", attn_high[idx1], attn_low[idx1], a_high[idx2]).
                                                        attn_high[idx1] = a_high[idx2];
                                                        trim = 0;
                                                }
                                                if ((attn_high[idx1] >= a_low[idx2]) && (attn_low[idx1] <= a_low[idx2])) {
//                                                      printf ("increase low %f %f %f\n", attn_high[idx1], attn_low[idx1], a_low[idx2]);
                                                        attn_low[idx1] = a_low[idx2];
                                                        trim = 0;
                                                }
```

B5

```
                                        if ((attn_high[idx1] <= a_high[idx2]) && (attn_low[idx1] >
a_low[idx2])) {
//                                              printf ("hold %f %f\n", attn_high[idx1],
attn_low[idx1]);
                                                trim = 0;
                                        }
                                }
                                if (attn_high[idx1] < attn_low[idx1]) {
                                        trim = 1;
                                }
                                if (trim) {
//                                      printf ("trim %f %f\n", attn_high[idx1], attn_low[idx1]);
                                        for (idx2 = idx1; idx2 < (count - 1); ++idx2) {
                                                attn_high[idx2] = attn_high[idx2+1];
                                                attn_low[idx2] = attn_low[idx2+1];
                                        }
                                        --count;
                                        --idx1;
                                }
                        }
                }
                if (count <= 0) {
//                      printf ("count break\n");
                        break;
                }
//              if (idx3 >= 10) break;
        }
        if (count > 1) {
                attn = (attn_high[0] + attn_low[0]) / 2.0;
                diff = attn_high[0] - attn_low[0];
                for (idx1 = 1; idx1 < count; ++idx1) {
                        if ((attn_high[idx1] - attn_low[idx1]) < diff) {
                                attn = (attn_high[idx1] + attn_low[idx1]) / 2.0;
                                diff = attn_high[idx1] - attn_low[idx1];
                        }
                }
//v             printf ("Muplitple potential attenuations detected, resolve by generating and matching\n"
//v                     "duplicated codes to those observed.  Currently choosing largest
range.\n");
        }
        else {
                attn = (attn_high[0] + attn_low[0]) / 2.0;
        }
        attn = log10 (attn) * 20.0;

//      printf ("count %d attempts %d range %f %f\n" count, idx3, attn_high[0], attn_low[0]);

//      printf ("attenuation detected %f\n", attn);

        return attn;
}

/* ***************************************************************** */

sint15
u_law_map_symbol_set (struct law_s *tx_law, struct law_s *rx_law)
{
        sint15 idx, count, limit;
        uint16 mask;

        total_symbols_per_frame.lsw = 1;
        total_symbols_per_frame.mid = 0;
        total_symbols_per_frame.msw = 0;

//      printf("%04x %04x %04x\n", total_symbols_per_frame.msw,
//              total_symbols_per_frame.mid, total_symbols_per_frame.lsw);

        for (idx = 0; idx < MAP_FRAME_SIZE; ++idx) {
                frame_slot_modulus[idx].lsw = total_symbols_per_frame.lsw;
                frame_slot_modulus[idx].mid = total_symbols_per_frame.mid;
                frame_slot_modulus[idx].msw = total_symbols_per_frame.msw;

#ifdef USE_4D_TRELLIS
                limit = tx_law->count;
                if (idx & 1) {
                        limit = limit >> 1;
                }
```

B6

```
                for (count = 1; count < limit; ++count) {
                        uint48_add (&total_symbols_per_frame, &frame_slot_modulus[idx],
                                        &total_symbols_per_frame);
                }
#else
                for (count = 1; count < tx_law->count; ++count) {
                        uint48_add (&total_symbols_per_frame, &frame_slot_modulus[idx],
                                        &total_symbols_per_frame);
                }
#endif
//              printf("%04x %04x %04x\n", total_symbols_per_frame.msw,
//                      total_symbols_per_frame.mid, total_symbols_per_frame.lsw);
        }

        count = 47;

        mask = 0x8000;
        for (idx = 0; idx < 16; ++idx) {
                if (mask & total_symbols_per_frame.msw) {
                        return count;
                }
                mask = mask >> 1;
                --count;
        }

        mask = 0x8000;
        for (idx = 0; idx < 16; ++idx) {
                if (mask & total_symbols_per_frame.mid) {
                        return count;
                }
                mask = mask >> 1;
                --count;
        }

        mask = 0x8000;
        for (idx = 0; idx < 16; ++idx) {
                if (mask & total_symbols_per_frame.lsw) {
                        return count;
                }
                mask = mask >> 1;
                --count;
        }
        return count;
}
/* ****************************************************************** */

void
print_deselect (sint15 deselect)
{
        if (deselect & LAW_DESELECTED_MIN) {
                printf ("(Below Min) ");
        }

        if (deselect & LAW_DESELECTED_MAX) {
                printf ("(Above Max) ");
        }

        if (deselect & LAW_DESELECTED_AVOID) {
                printf ("(Avoid Dmin) ");
        }

        if (deselect & LAW_DESELECTED_DMIN) {
                printf ("(Between Dmin) ");
        }

        if (deselect & LAW_DESELECTED_DUPLICATE) {
                printf ("(Duplicated Code) ");
        }

        if (deselect & LAW_DESELECTED_POWER) {
                printf ("(Power Limit) ");
        }

        printf("\n");
}
/* ****************************************************************** */
```

B7

```
//#define DO_FEW

/* ******************************************************************** */

//#define DO_PRINT_0
//#define DO_PRINT_1
//#define DO_PRINT_2
//#define DO_PRINT_3
//#define DO_PRINT_4
//#define DO_PRINT_5

/* ******************************************************************** */

#ifdef DO_FEW
#define MAX_ATTN 1
#define MAX_DMIN 4

#define START_DMIN  48
#define STEP_DMIN   4

#define START_ATTN  11.0
#define STEP_ATTN   1.0

/* ******************************************************************** */

#else /* DO_FEW */
#define MAX_ATTN 14
#define MAX_DMIN 36

#define START_DMIN  4
#define STEP_DMIN   4

#define START_ATTN  0.0
#define STEP_ATTN   1.0
#endif /* DO_FEW */

/* ******************************************************************** */

void
main (void)
{
        sint15 idx, count;
        float attn;
        sint15 bits_per_frame;
        sint31 bit_rate, overall_bit_rate;

        float input_attn;
        sint15 input_tx_dmin, input_rx_dmin;

        sint15 i, j, results_idx;

        struct results_s {
                float attn;
                sint15 dmin;
                sint31 bit_rate_simple;
                sint31 bit_rate_exclude;
                sint31 bit_rate_distinguishable_tx_dmin;
                sint31 bit_rate_distinguishable_rx_dmin;
        } results [MAX_ATTN * MAX_DMIN];

        results_idx = 0;

        input_attn = START_ATTN;
//      input_tx_dmin = 120;
//      input_rx_dmin = 120;

for (i = 0; i < MAX_ATTN; input_attn += STEP_ATTN, i++) {
        input_tx_dmin = START_DMIN;
        input_rx_dmin = START_DMIN;

for (j = 0; j < MAX_DMIN; input_tx_dmin += STEP_DMIN, input_rx_dmin += STEP_DMIN, j++) {

        printf ("\n");
        u_law_init (&u_law);

//      u_law_range (&u_law, 30, 4000);
        u_law_dmin (&u_law, input_tx_dmin);
//      u_law_power (&u_law, -12.0 /*-9.47981*/);
        u_law_power (&u_law, -12.0);
```

B8

```
        u_law_bit_rate (&u_law);

        printf ("%6.2f %3d %3d ", input_attn, input_tx_dmin, input_rx_dmin);

//      printf ("tx power %3.2f rx power %3.2f count %d dmin %d dup %d bit rate %ld\n",
        printf ("tx pwr %3.2f rx pwr %3.2f cnt %2d dmin %3d dup %2d rate %ld\n",
                        u_law.tx_power_db, u_law.rx_power_db, u_law.count, u_law.dmin,
                        u_law.indistinguishable, u_law.bit_rate);

#ifdef DO_PRINT_0
        for (idx = 0; idx < 128; ++idx) {
                printf ("%3d  %5d  %5d  %5d  %5d  ", u_law.value[idx].code, u_law.value[idx].linear,
                        u_law.value[idx].other, u_law_pcm_encode(u_law.value[idx].linear),
                        u_law_pcm_encode(u_law.value[idx].other - 1));
                print_deselect(u_law.value[idx].deselect);
        }
#endif

/* ****************************************************************** */

        u_law_attenuate (&u_law_rx, input_attn);
//      u_law_range (&u_law_rx, 22, 4000);
//      u_law_dmin (&u_law_rx, input_rx_dmin);
        u_law_attenuate_dmin (&u_law, &u_law_rx)      /* transfer from transmitter */
//      u_law_exclude (&u_law_rx);
//      u_law_discard_indinstinguishable (&u_law_rx);
        u_law_power (&u_law_rx, -12.0);
        u_law_discard_indinstinguishable (&u_law_rx);
        u_law_count_indinstinguishable (&u_law_rx);
        u_law_bit_rate (&u_law_rx);

        printf ("%6.2f %3d %3d ", input_attn, input_tx_dmin, input_rx_dmin);

//      printf ("tx power %3.2f rx power %3.2f count %d dmin %d dup %d bit rate %ld\n",
        printf ("tx pwr %3.2f rx pwr %3.2f cnt %2d dmin %3d dup %2d rate %ld\n",
                        u_law_rx.tx_power_db, u_law_rx.rx_power_db, u_law_rx.count,
                        u_law_rx.dmin, u_law_rx.indistinguishable, u_law_rx.bit_rate);

#ifdef DO_PRINT_1
        for (idx = 0; idx < 128; ++idx) {
                printf ("%3d  %5d  %5d  %5d  ", u_law.value[idx].code, u_law_rx.value[idx].other,
                        u_law_rx.value[idx].linear, u_law_rx.value[idx].code);
                print_deselect(u_law_rx.value[idx].deselect);
        }

        attn = u_law_detect_attenuation (&u_law_rx);
        printf ("attenuation detected  %f\n", attn);
#endif

        results[results_idx].attn = input_attn;
        results[results_idx].dmin = input_tx_dmin;
        if (u_law_rx.dmin == 0) {
                results[results_idx].bit_rate_simple = 0;
        }
        else {
                results[results_idx].bit_rate_simple = u_law_rx.bit_rate;
        }

/* ****************************************************************** */

        u_law_attenuate (&u_law_rx, input_attn);
//      u_law_range (&u_law_rx, 22, 4000);
//      u_law_dmin (&u_law_rx, input_rx_dmin);
        u_law_attenuate_dmin (&u_law, &u_law_rx);     /* transfer from transmitter */
        u_law_exclude (&u_law_rx);
//      u_law_discard_indinstinguishable (&u_law_rx);
        u_law_power (&u_law_rx, -12.0);
        u_law_discard_indinstinguishable (&u_law_rx);
        u_law_count_indinstinguishable (&u_law_rx);
        u_law_bit_rate (&u_law_rx);

        printf ("%6.2f %3d %3d ", input_attn, input_tx_dmin, input_rx_dmin);

//      printf ("tx power %3.2f rx power %3.2f count %d dmin %d dup %d bit rate %ld\n",
        printf ("tx pwr %3.2f rx pwr %3.2f cnt %2d dmin %3d dup %2d rate %ld\n",
                        u_law_rx.tx_power_db, u_law_rx.rx_power_db, u_law_rx.count,
                        u_law_rx.dmin, u_law_rx.indistinguishable, u_law_rx.bit_rate);

#ifdef DO_PRINT_2
```

B9

```
        for (idx = 0; idx < 128; ++idx) {
                printf ("%3d  %5d  %5d  %5d  ", u_law.value[idx].code, u_law_rx.value[idx].other,
                                u_law_rx.value[idx].linear, u_law_rx.value[idx].code);
                print_deselect(u_law_rx.value[idx].deselect);
        }

        attn = u_law_detect_attenuation (&u_law_rx);
        printf ("attenuation detected  %f\n", attn);
#endif

        if (u_law_rx.dmin == 0) {
                results[results_idx].bit_rate_exclude = 0;
        }
        else {
                results[results_idx].bit_rate_exclude = u_law_rx.bit_rate;
        }
/* ****************************************************************** */

        u_law_attenuate (&u_law_rx, input_attn);
//      u_law_range (&u_law_rx, 22, 4000);
//      u_law_dmin (&u_law_rx, input_rx_dmin);
        u_law_attenuate_dmin (&u_law, &u_law_rx);
        u_law_discard_indinstinguishable (&u_law_rx);
        u_law_power (&u_law_rx, -12.0);
//      u_law_power (&u_law_rx, (-12.0 - attn));
//      u_law_count_indinstinguishable (&u_law_rx);
        u_law_bit_rate (&u_law_rx);

        printf ("%6.2f %3d %3d ", input_attn, input_tx_dmin, input_rx_dmin);

//      printf ("tx power %3.2f rx power %3.2f count %d dmin %d dup %d bit rate %ld\n",
        printf ("tx pwr %3.2f rx pwr %3.2f cnt %2d dmin %3d dup %2d rate %ld\n",
                        u_law_rx.tx_power_db, u_law_rx.rx_power_db, u_law_rx.count,
                        u_law_rx.dmin, u_law_rx.indistinguishable, u_law_rx.bit_rate);

#ifdef DO_PRINT_3
        for (idx = 0; idx < 128; ++idx) {
                printf ("%3d  %5d  %5d  %5d  ", u_law.value[idx].code, u_law_rx.value[idx].other,
                                u_law_rx.value[idx].linear, u_law_rx.value[idx].code);
                print_deselect(u_law_rx.value[idx].deselect);
        }

        attn = u_law_detect_attenuation (&u_law_rx);
        printf ("attenuation detected  %f\n", attn);
#endif

        results[results_idx].bit_rate_distinguishable_tx_dmin = u_law_rx.bit_rate;
/* ****************************************************************** */

        u_law_attenuate (&u_law_rx, input_attn);
//      u_law_range (&u_law_rx, 22, 4000);
        u_law_dmin (&u_law_rx, input_rx_dmin);
//      u_law_attenuate_dmin (&u_law, &u_law_rx);
        u_law_discard_indinstinguishable (&u_law_rx);
        u_law_power (&u_law_rx, -12.0);
//      u_law_power (&u_law_rx, (-12.0 - attn));
//      u_law_count_indinstinguishable (&u_law_rx);
        u_law_bit_rate (&u_law_rx);

        printf ("%6.2f %3d %3d ", input_attn, input_tx_dmin, input_rx_dmin);

//      printf ("tx power %3.2f rx power %3.2f count %d dmin %d dup %d bit rate %ld\n",
        printf ("tx pwr %3.2f rx pwr %3.2f cnt %2d dmin %3d dup %2d rate %ld\n",
                        u_law_rx.tx_power_db, u_law_rx.rx_power_db, u_law_rx.count,
                        u_law_rx.dmin, u_law_rx.indistinguishable, u_law_rx.bit_rate);

#ifdef DO_PRINT_4
        for (idx = 0; idx < 128; ++idx) {
                printf ("%3d  %5d  %5d  %5d  ", u_law.value[idx].code, u_law_rx.value[idx].other,
                                u_law_rx.value[idx].linear, u_law_rx.value[idx].code);
                print_deselect(u_law_rx.value[idx].deselect);
        }

        attn = u_law_detect_attenuation (&u_law_rx);
        printf ("attenuation detected  %f\n", attn);
#endif
```

B10

```
        results[results_idx++].bit_rate_distinguishable_rx_dmin = u_law_rx.bit_rate;
/* ******************************************************************** */

        bits_per_frame = u_law_map_symbol_set (&u_law_rx, &u_law_rx);

        bit_rate = (sint31) (((float) (bits_per_frame) / 6.0 + 1.0) * 8000.0);

#ifdef USE_4D_TRELLIS
        overall_bit_rate = (sint31) (((float) (bits_per_frame) / 6.0 + 1.0 + 0.5) * 8000.0);
#else
        overall_bit_rate = (sint31) (((float) (bits_per_frame) / 6.0 + 1.0) * 8000.0);
#endif

#ifdef DO_PRINT_5
        printf ("bits per frame %d bit rate %ld overall bit rate %ld (with trellis)\n", bits_per_frame,
                        bit_rate, overall_bit_rate);
#endif

/* ******************************************************************** */

#if 0
        for (count = 0; count < 1000; count = count + bits_per_frame) {
                generate_bits (bits_per_frame);
                u_law_send_frame (&u_law, &u_law_rx);
                u_law_receive_frame (&u_law, &u_law_rx);
                check_bits (bits_per_frame);
        }
#endif

/* ******************************************************************** */
}
}

/* ******************************************************************** */

#ifndef DO_FEW
        for (idx = 0; idx < MAX_ATTN * MAX_DMIN; idx++) {
                printf("%6.2f %3d %5ld %5ld %5ld %5ld\n", results[idx].attn, results[idx].dmin,
                results[idx].bit_rate_simple, results[idx].bit_rate_exclude,
                results[idx].bit_rate_distinguishable_tx_dmin,
                results[idx].bit_rate_distinguishable_rx_dmin);
        }
#endif

}

/* ******************************************************************** */
```

B11

```
/* ********************************************************************** */
/*
 *      pcmcnv.c
 *
 *      (C) 1997 VoCAL Technologies Ltd.
 *
 *      ALL RIGHTS RESERVED.  PROPRIETARY AND CONFIDENTIAL.
 *
 *      VoCAL Technologies Ltd.
 *      3032 Scott Blvd.
 *      Santa Clara, CA 95054
 *
 *      Product:        C
 *
 *      Module:                 PCM
 *
 *      This file contains the PCM conversion functions.
 *
 *      Revision Number:        $RevisionS
 *      Revision Status:        $State$
 *      Last Modified:          $Date$
 *      Identification:         $Id$
 *
 *      Revision History:       $Log$
 *      Revision 1.0  1997/03/01  00:00:00  VD
 *      Initial release of software
 *
 */
/* ********************************************************************** */

#include "standard.h"
#include "pcm.h"

/* ********************************************************************** */

#include <stdio.h>

sint15
u_law_pcm_encode (sfract15 input)
{
        sfract15 value, seg;

        value = input;                          /* Take absolute value */
        if (input < 0) value = -input;

        value = value + 33;                     /* Limit values */
        if (value > 8159) value = 8159;

        seg = iexp (value);                     /* Compute exponent */
//printf("%5d  %5d  %5d", input, value, seg);
        value = norm (value, seg);          /* Normalize value */

        value = value ^ 0x4000;             /* Clear bit */
        value = value >> 10;                /* Position bits */
//printf("  %5d", value);

        if (input < 0) {
                value = value | 0x0080;     /* Insert sign bit */
        }

        seg = seg + 9;                          /* Compute segment */
        seg = seg << 4;

        value = value | seg;                /* Insert segment bits */
//printf("  %04x", value);

        value = value ^ 0x00ff;             /* Invert bits */
//printf("  %5d\n", value);
        return (sint15) value;
}

#ifdef DO_U_LAW
x       :                               /* Input sample passed in AR (1.15) */
x                       AR = ABS AR;            /* Take absolute value */
x                       AY0 = 132;                  /* 33 << 2 */
x                       AR = AR + AY0;
x                       SR1 = 32636;            /* Limit values */
x                       IF AV AR = PASS SR1;
x                       SE = EXP AR (HI);       /* Compute exponent */
```

B12

```
x                   SR = NORM AR (LO);              /* Normalize value */
x                   AY0 = 0x4000;
x                   AR = SR0 XOR AY0;               /* Clear bit */
x                   SR = LSHIFT AR BY -10 (LO);     /* Position bits */
x                   AX0 = SE, AR = PASS AY0;
x                   IF POS AR = PASS 0;             /* Create sign bit */
x                   SR = SR OR LSHIFT AR BY -7 (LO);     /* Insert sign bit */
x                   AY0 = 7;                             /* 9 if using 8031 range */
x                   AR = AX0 + AY0;                      /* Compute segment */
x                   SR = SR OR LSHIFT AR BY 4 (LO);      /* Insert segment bits */
x                   AY0 = 0xFF;
x                   AR = SR0 XOR AY0;               /* Invert bits */
x                   RTS;                                 /* Pass on rate converted value *
#endif

/* ******************************************************************** */

sfract15
u_law_pcm_decode (uint8 code)
{
        sfract15 value, seg;

        value = code & 0x00ff;                  /* Mask bits */
        value = value ^ 0x00ff;                 /* Invert bits */

        seg = value & 0x0070;                   /* Isolate segment bits */
        value = value ^ seg;                    /* Remove segment bits */
        seg = seg >> 4;                              /* Shift to low bits */

        if ((value - 0x0080) >= 0) {        /* Determine sign */
                value = value + 0xff80          /* Process negative value */
                value = value + value + 33;     /* Add segment offset */
                value = value << seg;           /* Position bits */
                value = 33 - value;                  /* Remove segment offset */
        }
        else {                                            /* Process positive value */
                value = value + value + 33;     /* Add segment offset */
                value = value << seg;           /* Position bits */
                value = value - 33;                  /* Remove segment offset */
        }

        return value;
}
/* ******************************************************************** */

sfract15
u_law_pcm_threshold (uint8 code)
{
        sfract15 value, seg;

        value = code & 0x00ff;                  /* Mask bits */
        value = value ^ 0x00ff;                 /* Invert bits */

        seg = value & 0x0070;                   /* Isolate segment bits */
        value = value & 0x000f;                 /* Remove segment bits */
        seg = seg >> 4;                              /* Shift to low bits */
        seg++;

        value = value + 17;                          /* Add segment offset */
        value = value << seg;                   /* Position bits */
        value = value - 33;                          /* Remove segment offset */

        if ((code & 0x80) == 0) {           /* Determine sign */
                value = -value;                      /* Process negative value */
        }

        return value;
}
/* ******************************************************************** */

#ifdef DO_A_LAW

a_law_pcm_encode:                                        /* Input sample passed in AR (1.15) */
                   AR = ABS AR;                          /* Take absolute value */
                   AY0 = 511;       /* 127 */            /* Check for zero segment */
                   AF = AR - AY0;
                   IF GT JUMP a_law_pcm_enc_1;
```

B13

```
                SR = LSHIFT AR BY -4 (LO).    /* Downshift bits */
                AR = 0x4000;
                IF NEG AR = PASS 0;           /* Create sign bit */
                SR = SR OR LSHIFT AR BY -7 (LO);    /* Insert sign bit */
                AY0 = 0x55;
                AR = SR0 XOR AY0;             /* Invert bits */
                RTS;                                /* Pass on rate converted value */

a_law_pcm_enc_1:
                SE = EXP AR (HI).             /* Compute exponent */
                AX0 = SE, SR = NORM AR (LO);        /* Normalize value */
                AY0 = 0x4000;
                AR = SR0 XOR AY0;             /* Clear bit */
                SR = LSHIFT AR BY -10 (LO);   /* Position bits */
                AR = PASS AY0;
                IF NEG AR = PASS 0;           /* Create sign bit */
                SR = SR OR LSHIFT AR BY -7 (LO);    /* Insert sign bit */
                AY0 = 7;
                AR = AX0 + AY0;                     /* Compute segment */
                IF LT AR = PASS 0;
                SR = SR OR LSHIFT AR BY 4 (LO);     /* Insert segment bits */
                AY0 = 0x55;
                AR = SR0 XOR AY0;             /* Invert bits */
                RTS;                                /* Pass on rate converted value */

                ident(IDENT_ RX_B1_ PCM_DECODE);

a_law_pcm_decode:                             /* Input sample passed in AR */
                AY0 = 0x55,                         /* Mask bits for inversion */
                AR = AR XOR AY0;              /* Invert bits */
                SR1 = 0x8;                          /* Set sign bit */
                SR0 = 0x800;                  /* Set LSB of interval */
                SR = SR OR LSHIFT AR BY -12 (LO);   /* Isolate segment and interval */
                AY0 = 32;
                AX0 = AR. AF = PASS AY0;
                AY0 = 9;                            /* Segment bias */
                AR = SR1 - AY0;                     /* Determine shift */
                IF LT AF = PASS 0;            /* No extra MSB bit */
                IF EQ AR = PASS 0;            /* No less than zero bits */
                SR = LSHIFT SR0 BY -11 (LO); /* Isolate interval */
                SE = AR, AR = SR0 OR AF;      /* Add bit if necessary */
                SR = LSHIFT AR (LO); /* Position output */
                SR = LSHIFT SR0 BY 3 (LO);
                AY0 = 0xFF80;
                AR = SR0, AF = AX0 + AY0;     /* Check if sign bit set */
                IF LT AR = - SR0;             /* If it is, negate result */
                RTS;                                /* Pass on rate converted value (1.15) */
#endif /* DO_A_LAW */
```

B14

```
/* ********************************************************************** */
/*
 *    math.c
 *
 *    (C) 1997 VoCAL Technologies Ltd.
 *
 *    ALL RIGHTS RESERVED.  PROPRIETARY AND CONFIDENTIAL.
 *
 *    VoCAL Technologies Ltd.
 *    3032 Scott Blvd.
 *    Santa Clara, CA 95054
 *
 *    Product.       C
 *
 *    Module.        MATH
 *
 *    This file contains fractional math support functions
 *
 *    Revision Number:      $Revision$
 *    Revision Status:      $State$
 *    Last Modified:        $Date$
 *    Identification:       $Id$
 *
 *    Revision History.     $Log$
 *    Revision 1.0  1997/03/01  00.00:00  VD
 *    Initial release of software
 *
 */
/* ********************************************************************** */

#include "standard.h"
#include "pcm.h"
#include <math.h>

/* ********************************************************************** */

sint15
iexp (sfract15 value)
{
      sint15 exp.

      if (value < 0) {
            value = -value;
      }

      exp = 15;

      if (value == 0) {
            return exp;
      }

      exp = 0;
      while ((value & 0x4000) == 0) {
            --exp;
            value = value << 1;
      }

      return exp
}
/* ********************************************************************** */

sint15
norm (sfract15 value, sint15 exp)
{
      return (value << -exp);
}
/* ********************************************************************** */

float
db_to_float (float db)
{
      float db_float;

      db_float = db / 20.0;
      db_float = pow (10.0, db_float);

      return db_float;
```

B15

```
}
/* ******************************************************************* */

float
db_to_power (float db)
{
      float db_float;

      db_float = db / 10.0;
      db_float = pow (10.0, db_float);

      return db_float;
}
/* ******************************************************************* */

float
float_to_db (float power)
{
      float float_db;

      float_db = log10 (power) * 20.0;

      return float_db;
}
/* ******************************************************************* */

float
power_to_db (float power)
{
      float power_db;

      power_db = log10 (power) * 10.0;

      return power_db;
}
/* ******************************************************************* */

void
uint48_add (uint48 *src1, uint48 *src2, uint48 *dst)
{
      uint16 s1l, s1m, s1h, s2l, s2m, s2h, dstl, dstm, dsth;

      s1l = src1->lsw;
      s1m = src1->mid;
      s1h = src1->msw;

      s2l = src2->lsw;
      s2m = src2->mid;
      s2h = src2->msw;

      asm {
            mov ax, s1l;
            add ax, s2l;
            mov dstl, ax;

            mov ax, s1m;
            adc ax, s2m;
            mov dstm, ax;

            mov ax, s1h;
            adc ax, s2h;
            mov dsth, ax;
      }

      dst->lsw = dstl;
      dst->mid = dstm;
      dst->msw = dsth;

//    printf("%04x %04x %04x = %04x %04x %04x + %04x %04x %04x\n",
//                dsth, dstm, dstl, s1h, s1m, s1l, s2h, s2m, s2l);
}
/* ******************************************************************* */

void
```

B16

```
uint48_sub (uint48 *src1, uint48 *src2, uint48 *dst)
{
     uint16 s1l, s1m, s1h, s2l, s2m, s2h, dstl, dstm, dsth;

     s1l = src1->lsw;
     s1m = src1->mid,
     s1h = src1->msw;

     s2l = src2->lsw;
     s2m = src2->mid,
     s2h = src2->msw;

     asm {
           mov ax  s1l;
           sub ax, s2l;
           mov dstl, ax;

           mov ax, s1m;
           sbb ax, s2m;
           mov dstm, ax;

           mov ax, s1h;
           sbb ax, s2h;
           mov dsth, ax;
     }

     dst->lsw = dstl;
     dst->mid = dstm;
     dst->msw = dsth;
}
/* ********************************************************************** */

sfract15
fmpy (sfract15 a, sfract15 b)
{
     long a_long, b_long;
     short a_short;

     a_long = a;
     b_long = b;

     a_long = a_long * b_long;
     a_short = (short) (a_long >> 15);

     return a_short;
}
/* ********************************************************************** */
```

B17

```
/* ******************************************************************** */
/*
 *      pcm.h
 *
 *      (C) 1997 VoCAL Technologies Ltd.
 *
 *      ALL RIGHTS RESERVED.  PROPRIETARY AND CONFIDENTIAL.
 *
 *      VoCAL Technologies Ltd.
 *      3032 Scott Blvd
 *      Santa Clara, CA  95054
 *
 *      Product:        C
 *
 *      Module:                 PCM
 *
 *      This file defines the PCM utility functions.
 *
 *      Revision Number:        $Revision$
 *      Revision Status:        $State$
 *      Last Modified:          $Date$
 *      Identification:         $Id$
 *
 *      Revision History:       $Log$
 *
 */
/* ******************************************************************** */

#ifndef  _PCM_PCM_H
#define  _PCM_PCM_H

/* ******************************************************************** */

typedef struct {
        uint16 lsw;
        uint16 mid;
        uint16 msw;

} uint48;

sint15 iexp (sfract15 value);
sint15 norm (sfract15 value, sint15 exp);
float db_to_float (float db);
float db_to_power (float db);
float float_to_db (float power);
float power_to_db (float power);

void uint48_add (uint48 *src1, uint48 *src2, uint48 *dst);
void uint48_sub (uint48 *src1, uint48 *src2, uint48 *dst);

sint15 u_law_pcm_encode (sfract15 input);
sfract15 u_law_pcm_decode (uint8 code);
sfract15 u_law_pcm_threshold (uint8 code);

/* ******************************************************************** */

#endif /* _PCM_PCM_H */
```

B18

```
/* ******************************************************************* */
/*
 *      standard.h
 *
 *      (C) 1994, 1995, 1996 VoCAL Technologies Ltd.
 *
 *      ALL RIGHTS RESERVED. PROPRIETARY AND CONFIDENTIAL.
 *
 *      VoCAL Technologies Ltd.
 *      3032 Scott Blvd
 *      Santa Clara, CA  95054
 *
 *      Product:        MODEM 101
 *
 *      Module:                 SYSTEM
 *
 *      This file defines the standard system definitions.
 *
 *      Revision Number         $Revision$
 *      Revision Status:        $State$
 *      Last Modified:          $Date$
 *      Identification:         $Id$
 *
 *      Revision History:       $Log$
 *
 */
/* ******************************************************************* */

#ifndef _SYSTEM_STANDARD_H
#define _SYSTEM_STANDARD_H

/* ******************************************************************* */

#ifdef DO_CX025
#include <GLBL_Globals.h>
#include <m68EC040.h>
#endif /* DO_CX025 */

#ifdef DO_WIN

/*-#include <_null.h>-*/
#define __STDC__ FALSE
//#include <stddef.h>

#define strupr(a) _strupr(a)
#define strlwr(a) _strlwr(a)
#define inp(a) _inp(a)
#define inpw(a) _inpw(a)
#define outp(a,b) _outp(a,b)
#define outpw(a,b) _outpw(a,b)

#ifdef DO_WIN32
#ifdef DO_ADI2181
#define DEBLEVEL 1
#define DEBUG
#include <debug.h>                              /*- DDK Debug Header File -*/
//#define _Debug_Printf_Service LCODE__Debug_Printf_Service
//#include <vxdwraps.h>
#endif /* DO_ADI2181 */
#define far
#define near
#define NULL  ((void *)0)
#endif /* DO_WIN32 */

#else /* DO_WIN */

#ifndef DO_CX018__A
/*#include <_null.h>*/
#ifndef NULL
#if defined(__TINY__) || defined(__SMALL__) || defined(__MEDIUM__)
#define NULL   0
#else
#define NULL   0L
#endif
#endif
#endif /* DO_CX018__A */

#endif /* DO_WIN */
```

B19

```
/* ******************************************************************* */

typedef unsigned char uint8;      /* range 0-255 */
typedef unsigned char umod8:  /* 8 bit math requirements */
typedef unsigned char octet:      /* octet data */
#ifdef DO_FORCED__BYTE_ALIGNED
typedef unsigned short pack8;
#else /* DO_FORCED__BYTE_ALIGNED */
typedef unsigned char pack8;    /* packed octet data */
#endif /* DO_FORCED__BYTE_ALIGNED */
typedef pack8 expand8; /* expand octet data to uint16 */
typedef signed char sint7;

typedef unsigned short uint16;
typedef signed short sint15,
typedef signed short sfract15,

typedef unsigned long uint32.
typedef signed long sint31.

#ifdef DO_CX025
#define far
#define interrupt
#endif /* DO_CX025 */

/* ******************************************************************* */

#define MAX_UINT8 (255)
#define MAX_SINT7 (127)
#define MIN_SINT7 (-128)

#define MAX_UINT16 (65535)
#define MAX_SINT15 (32767)
#define MIN_SINT15 (-32768)

#define MAX_UINT32 (4294967296)
#define MAX_SINT31 (2147483647)
#define MIN_SINT31 (-2147483648)

#define MIN_DOUBLE (1.7e-308)
#define MAX_DOUBLE (1.7e308)

/* ******************************************************************* */

typedef uint8 byte;
typedef uint16 word;

typedef int bool;

#ifndef DO_CX025
#define FALSE 0
#define TRUE (!FALSE)
#endif /* DO_CX025 */

#if defined(DO_ADI2181) || defined(DO_ISAR)
typedef uint32 dsp_pm_t;       /* Of size of DSP program memory contents    */
typedef uint16 dsp_dm_t;       /* Of size of DSP data memory contents       */
typedef uint16 dsp_addr_t;     /* Of size of DSP addresses                                */
typedef uint16 dsp_xddr_t;     /* Of size of extended DSP addresses         */
#endif /* DO_ADI2181 || DO_ISAR */

/* ******************************************************************* */

#define forever for (;;)

#ifdef DO_CX025
#define interrupt_disable_onto_stack() \
                        asm( "ori.w    #0x0700, sr")
#define interrupt_restore_from_stack() \
                        asm( "andi.w    #0xF8FF, sr")
#else /* DO_CX025 */

#ifdef DO_CX018__A
#define interrupt_disable_onto_stack() \
  {asm(" ORI #$700,SR\n NOP\n NOP\n");}
#define interrupt_restore_from_stack() \
  {asm(" ANDI #$f8ff,SR\n NOP\n NOP\n");}
#else /* DO_CX018__A */

#ifdef DO_WIN
#define interrupt_disable_onto_stack()
```

B20

```
#define interrupt_restore_from_stack()
#else /* DO_WIN */
#define interrupt_disable_onto_stack() {asm pushf;asm cli;}
#define interrupt_restore_from_stack() {asm popf;}
#endif /* DO_WIN */

#endif /* DO_CX018__A */
#endif /* DO_CX025 */

#ifdef DO_CAPI20
#define DEBLEVEL      1
#pragma code_seg("_LTEXT", "LCODE")
#include <debug.h>
#include <vxdwraps.h>

#else /* DO_CAPI20 */
#ifdef DO_ISAR
typedef const char far* LPCSTR;
typedef char far* LPSTR;
#define PASCAL pascal
#endif /* DO_ISAR */
#endif /* DO_CAPI20 */

/* ******************************************************************** */

#ifdef DO_WIN32
//#define sprintf _Sprintf
#endif /* DO_WIN32 */

/* ******************************************************************** */
/* ******************************************************************** */
/* ******************************************************************** */
/* ******************************************************************** */
/* ******************************************************************** */

#define print_debug_off(A)
#define print_diag_off(A)

#ifndef DO_CAPI20
#define print_primary(A) print_routine A
#define print_std(A) print_routine A
#define print_info(A) print_routine A
#define print_diag_on(A) print_routine A
#define print_debug_on(A) print_routine A
#define report_anomaly(A) report_anomaly_routine (A)
#else /* DO_CAPI20 */

#define print_std(A) Debug_Printf A
#define print_info(A) Debug_Printf A
#define print_diag_on(A) Debug_Printf A
#define print_debug_on(A) Debug_Printf A
#define report_anomaly(A) report_anomaly_routine (A)

#endif /* DO_CAPI20 */

#ifdef DO_PRINTS_DISABLED
#undef print_primary
#undef print_std
#undef print_info
#undef print_debug
#undef report_anomaly
#define print_primary(A)
#define print_std(A)
#define print_info(A)
#define print_debug(A)
#define report_anomaly(A)
#endif /* DO_PRINTS_DISABLED */

#ifdef DO_CX025
#undef print_primary(A)
#undef print_std(A)
#undef print_info(A)
#undef print_debug(A)
#undef report_anomaly(A)
#endif /* DO_CX025 */

/* ******************************************************************** */

#endif /* SYSTEM_STANDARD_H */
```

B21